СЕРГЕЙ ЗУЕВ
НИКОЛАЙ ПОЛЕЩУК

САПР на базе AutoCAD — как это делается

Стандарты проектирования и разработки

COM-технологии

Связь AutoCAD и Delphi

AutoCAD и базы данных

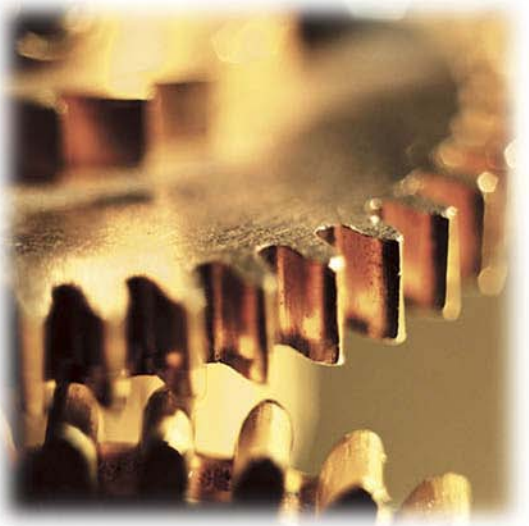
Иллюстрированные XML-меню

Интеграция всех разделов проекта

Инженерные расчеты

Элементы документооборота

Интеграция САПР и ГИС



МАСТЕР +CD-ROM

Сергей Зуев
Николай Полещук

САПР на базе AutoCAD — как это делается

Санкт-Петербург
«БХВ-Петербург»
2004

УДК 681.3.06
ББК 32.973.26-018.2
З-93

Зуев С. А., Полещук Н. Н.

З-93 САПР на базе AutoCAD — как это делается. — СПб.: БХВ-Петербург, 2004. — 1168 с.: ил.

ISBN 5-94157-344-8

В книге, основанной на собственном опыте авторов, их многочисленных ошибках и находках, раскрывается технология создания комплексной САПР, предназначенной для разработки различных чертежей — от идеи и концепции до выпуска продукта в свет. Разбирается, как и что можно и нужно делать без программирования, какие инструменты разработчика в системе AutoCAD (AutoLISP, Visual LISP, ObjectARX, VBA, Delphi) могут быть использованы, вырабатываются рациональные приемы программирования. Приводятся примеры прикладных программ различного назначения для конечных пользователей и для разработчиков САПР. Рассматриваются элементы документооборота и сочетания геоинформационных технологий с автоматизированным проектированием. Особое внимание уделяется нетрадиционным для AutoCAD технологиям — Delphi, COM-серверам, XML-меню. Основной упор сделан на то, чтобы показать, как это делается практически.

Для инженеров и программистов, занимающихся вопросами САПР

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Владимир Шабалин*
Зав. редакцией	*Григорий Добин*
Редактор	*Юрий Рожко*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 02.07.04.
Формат 70×100$^{1}/_{16}$. Печать офсетная. Усл. печ. л. 94,17.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Гигиеническое заключение на продукцию, товар № 77.99.02.953.Д.001537.03.02
от 13.03.2002 г. выдано Департаментом ГСЭН Минздрава России.

Отпечатано с готовых диапозитивов
в ОАО "Техническая книга"
190005, Санкт-Петербург, Измайловский пр., 29

ISBN 5-94157-344-8

Содержание

Введение

Наша книга называется "САПР на базе AutoCAD — как это делается". Все слова из названия этой книги являются ключевыми для понимания, о чем идет речь. САПР — это система автоматизированного проектирования. Именно о системе, а не об отдельных программах и приемах работы пойдет речь. Причем о системе, базирующейся на системе AutoCAD фирмы Autodesk (США). Система AutoCAD, одна из самых популярных в мире и наверняка самая популярная в России, является мощным графическим ядром, на котором базируются многие прикладные пакеты, как от самой фирмы Autodesk (Mechanical Desktop, Architectural Desktop, Land Development Desktop и т. п.), так и огромное количество программ и систем от партнеров Autodesk и независимых производителей. В дальнейшем в целях ясности "голую" систему AutoCAD будем называть базовой системой.

Сама система AutoCAD в "голом" виде, т. е. в объеме базовой поставки, позволяет выполнять чертежные (и не только чертежные) работы любого назначения с использованием только штатных средств. С помощью стандартных примитивов AutoCAD (отрезков, полилиний, кругов, текстов и т. п.) можно "нарисовать" почти все, что угодно. Но конечному пользователю (конструктору-машиностроителю, строителю, сантехнику) нужно рисовать не просто наборы примитивов, а конкретные объекты (здания, оборудование, трубопроводы, изделия, планы городов и многое другое). Делать это надо быстро и качественно, в соответствии со стандартами, действующими в той или иной стране и отрасли.

Все, что требуется всем конечным пользователям во всех странах, фирма Autodesk включить в состав системы AutoCAD не могла, да этого и не нужно делать, т. к. в этом случае получился бы некий программный монстр, в котором при наличии огромного количества средств конкретному человеку всегда чего-то бы не хватило, а все остальное использовалось бы только на несколько процентов.

Фирма Autodesk пошла, на наш взгляд, по правильному пути — она снабдила *базовую* систему превосходными средствами разработки *прикладных* систем. Используя средства разработки AutoCAD, можно создавать программы и для вычерчивания зданий, и для проектирования судов, и для "болтов и гаек".

А "как это делается" (крылатое выражение Карела Чапека), мы и попытаемся осветить в нашей книге.

Особенности книги

В СССР и России издано немало книг, посвященных и работе с системой AutoCAD на уровне обычного пользователя, и ее адаптации, и программированию на LISP (основном, или, по крайней мере, самом старом средстве для разработчиков). В этих книгах, как правило, изучается язык программирования, приводятся примеры разработки нескольких программ, публикуются справочные сведения по функциям, командам AutoCAD и его системным переменным. Обычно этого бывает достаточно, чтобы читатель смог начать, и весьма успешно, разработку собственных программ.

Однако после того как первые трудности с освоением языка и элементарных приемов работы остаются позади, возникают вопросы другого плана. Сначала смутно, а затем все яснее разработчик начинает понимать, что созданные им десятки (или даже сотни) программ надо приводить в какую-то *систему*. У него возникает желание распространять свои разработки и получать за это деньги. На этом этапе приходит понимание разницы между *программой* (даже очень хорошей) и *программным продуктом*. И чаще всего выясняется, что с самого начала многое надо было делать не так, многое требуется переписать заново. Обнаруживается, что программы почему-то работают только в присутствии автора, на его компьютере, приходится расставаться с мечтами о заработке на продажах.

В этой книге, основанной на собственном опыте, многочисленных ошибках и находках, мы попытаемся раскрыть технику и технологию разработки комплексной САПР, предназначенной для разработки различных чертежей — от идеи и концепции до выпуска продукта в свет. Причем сделаем это на конкретном примере гипотетической "Нашей САПР". Гипотетической, но собранной в виде реального дистрибутива (программы установки системы), который можно будет найти на сайтах авторов.

Вопросы, рассматривающиеся в книге, не "высосаны из пальца". Большинство из них возникало (и более или менее успешно решалось) в ходе разработки реальных прикладных систем. Кроме того, в течение ряда лет мы участвуем в работе нескольких форумов в Интернете, в той или иной мере связанных с разработкой приложений для AutoCAD. Анализ тысяч вопросов, задававшихся на этих форумах, помог включить в книгу ряд актуальных и интересных тем.

Эта книга не является справочником, учебником или методическим пособием для изучения AutoCAD или языков программирования. В ней вы не встретите ни перечней команд и системных переменных AutoCAD, ни исчерпывающих перечней функций, процедур и методов, используемых в рассматриваемых языках. Все это можно найти в других книгах, в том числе в книгах Николая Полещука[1]. Все это имеется и в справочной системе самого AutoCAD (увы, локализируемой в последнюю очередь). Сведения справочного характера приводятся только для официально не документированных функций, методов и переменных. В нашей книге мы постарались раскрыть такие темы, которые или вообще не рассматривались ранее, или очень смутно освещены в технической документации.

---

[1] Полещук Н. Н. Visual LISP и секреты адаптации AutoCAD. — СПб: БХВ-Петербург, 2001; Полещук Н. Н. AutoCAD 2002. — СПб: БХВ-Петербург, 2003; Полещук Н. Н. AutoCAD 2004. — СПб: БХВ-Петербург, 2004; Полещук Н. Н. AutoCAD 2004: разработка приложений и адаптация. — СПб: БХВ-Петербург, 2004.

Для кого предназначена эта книга

Эта книга позиционируется как издание, предназначенное для *разработчиков прикладных систем на базе AutoCAD*. Обратите внимание — не для *программистов* (хотя, разумеется, может быть прочитана и ими).

Следует отметить, что разработкой прикладных программ на базе AutoCAD занимаются, как правило, не "настоящие программисты", а обычные инженеры (по крайней мере, в России). Вызвано это, по нашим наблюдениям, тем, что проще инженеру освоить не очень сложные средства разработки для AutoCAD (и получать при этом великолепные результаты), чем заставить профессионального программиста заниматься этой "грязной" работой. Действительно, задачи, решаемые "под AutoCAD", обычно не требуют ни сложных математических методов, ни хитроумной обработки больших объемов данных, ни умелого обращения с ресурсами операционной системы. Зато решающее значение имеют, казалось бы, мелкие нюансы интерфейса пользователя с программой, знание предметной области, глубокое знание самого AutoCAD и, обязательно, *любовь к пользователю*. Все это приходит только в результате опыта практической работы по проектированию (как "бумажного", так и "компьютерного").

Зачастую российский инженер, сделав несколько программ "для себя", заболевает этой "неизлечимой болезнью" — САПР, и продолжает заниматься разработкой на все более профессиональном уровне. Чаще он продолжает одновременно заниматься проектированием или конструированием, иногда проходит путь до профессионального *разработчика*. Одной из задач этой книги является помощь в сокращении этой трудной дороги.

Наверное только в России такое количество людей занимается не своим делом. Архитекторы пишут музыку и учат с телеэкранов секретам приготовления пищи, авиационные инженеры сочиняют и читают со сцены юмористические рассказы, а инженеры-строители разрабатывают информационные системы, не имеющие к строительству никакого отношения (именно этим и занят сейчас один из авторов). А профессиональные программисты, не попавшие в команды, работающие над конкретными проектами, мучаются в поисках "что бы такого сделать плохого" (и делают), в то время как какой-нибудь конструктор мучительно пытается понять, как из AutoCAD добраться до сервера Interbase. Возможно, наша книга и наставит "на путь истинный" кого-то из программистов, и подскажет новые решения для конструкторов.

Структура книги

Мы постарались написать эту книгу так, чтобы ее конечная цель была достигнута с минимальными моральными и материальными потерями для читателей. Книга переполнена исходными текстами, читать которые лучше не за тарелкой с борщом, а за компьютером. Впрочем, не торопитесь "набивать" все встречающиеся "исходники". Очень может быть, что вы их найдете на прилагаемом диске, на сайте издательства или на сайтах авторов.

В книге рассматривается пример создания реальной САПР, которую читатель сможет использовать в своей работе. Но сделано это не в виде пошаговых инструкций

(хотя такие и встречаются для нетривиальных случаев). Разработать САПР "шаг за шагом, и только вперед" нельзя. Чаще придется делать "три шаги налево, три шаги направо, шаг вперед и два назад". Соответственно нельзя и написать книгу с последовательным изложением таких "шагов".

Книга состоит из нескольких частей:

- В *части I* рассматривается общая концепция создаваемой нами САПР. Мы продумаем принципы работы над проектом, разберем постановочные вопросы, определимся, какую же систему мы будем разрабатывать, какие задачи и как она будет решать, чем она будет отличаться от аналогов.
- В *части II* мы рассмотрим, как и что можно и нужно делать без программирования (библиотеки блоков, штриховок, типов линий), т. е. на уровне *адаптации* AutoCAD. Толково сформированные библиотеки в сочетании с простыми программами для их использования позволяют автоматизировать весьма значительную часть рутинных операций по отрисовке различных изображений.
- *Часть III* посвящена разработке инструментальных средств для прикладного программирования. В ней мы разберем различные инструменты разработчика в системе AutoCAD (AutoLISP, Visual LISP, ObjectARX, VBA, Delphi), выработаем рациональные приемы программирования и разработаем все требуемые библиотечные функции, облегчающие разработку прикладных программ.
- В *части IV* рассматриваются программы, с которыми и будет работать конечный пользователь. При этом мы вначале сформируем небольшой "сундучок программиста" с набором инструментов для своих корыстных целей — облегчения собственной работы, затем напишем десяток простеньких (в 2—3 строчки) программ общего назначения. Эти простые программы позволят сразу добавить в меню системы AutoCAD несколько сотен *команд* для создания нескольких тысяч изображений. А пока пользователи разбираются со свалившимися на них возможностями, мы напишем еще множество программ специального назначения — для строителей и сантехников, топографов и генпланистов, электриков и газовиков. Мы дополним нашу систему элементами документооборота — улучшим работу с папками и документами, сделаем простой, но пригодный для реальной работы электронный архив с кальками, типовыми решениями и различными документами.
- *Часть V*, последняя, рассказывает о том, как выпустить систему в свет, превратив большое количество программ в программный продукт. Создание справочной системы и документации, подготовка инсталляций и (почему бы и нет?) технологии распространения — все здесь.

Предупреждения

В этой книге рассматриваются многие темы, для которых просто не может быть однозначно верного и единственного решения (разве что перевод градусов в радианы и обратно). Проектирование вообще, а проектирование системы проектирования особенно — процесс творческий. Готовых рецептов здесь быть не может. Мы предлагаем только направления для размышлений и некоторые варианты решений. Очень может быть, что читатели найдут (непременно найдут) лучшие варианты — как на уровне концепции, так и в реализации. Что же, авторы будут этому только рады.

Из-за неоднозначности решения рассматриваемых вопросов единого мнения может не быть и у авторов книги. В таких случаях мы будем выделять соображения каждого автора в виде специальных врезок "особое мнение", предоставив читателям право на окончательный вердикт.

Хотя в книге встречаются некоторые "секретные материалы", не стоит думать, что в ней будут раскрыты абсолютно все наши и чужие секреты.

- Во-первых, все их мы не знаем.
- Во-вторых, надо что-то оставить и себе.
- В-третьих, и это самое главное: мы не скрываем, что очень многие сведения получаем из Интернета, а там самые любопытные материалы (например, по недокументированным возможностям AutoCAD) часто сопровождаются различными угрозами, в которых упоминаются некие "пещеры", "ритуалы", "злые духи", "блондинки" (причем подчеркивается, что именно самки, а не самцы), даются тонкие намеки на "визиты в четверг между полуднем и двумя часами" и прочие прелести. Не желая подвергать себя и читателей риску ненужных "визитов" и "ритуалов", мы постараемся воздержаться от публикации кое-чего. Пока ограничимся.

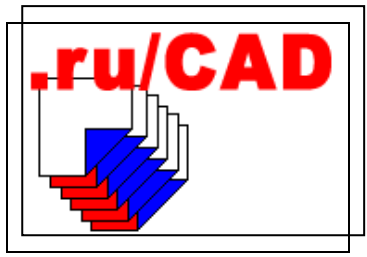
Об авторах

Кроме авторов, указанных на обложке, созданию этой книги в той или иной мере способствовали наши коллеги и единомышленники по любви к AutoCAD:

- Петр Лоскутов (г. Екатеринбург) — наш консультант, помощник, оппонент (все в одном лице), который давал ценные замечания по тексту книги, оптимизировал многие функции и предоставил свои разработки и исследования, оказавшие важное влияние на реализацию рассматриваемой системы.
- Александр Якушин (г. Курган) — вместе с ним один из авторов этой книги разрабатывал интегрированную систему BestIA, многие идеи и решения из которой рассмотрены в книге.
- Михаил Федоров (г. Курган) — реализовывал в исходных текстах и проверял некоторые безумные идеи, возникавшие в работе над книгой.
- Михаил Камнев (г. Курган) — консультировал авторов по работе с C++.
- Участники Web-конференции пользователей систем CAD/CAM/CAE/GIS (**http://www.cad.dp.ua/wboard/index.php**) — давали интересные ответы на вопросы, освещаемые в книге.
- Многочисленные участники форума сайта AutoCAD.ru (**http://www.autocad.ru/cgi-bin/f1/board.cgi?&action=show_tree**) — задавали самые разные вопросы, позволившие лучше определить круг интересов наших читателей.

В тексте книги при цитировании конкретных соавторов мы будем давать соответствующие ссылки.

ЧАСТЬ I



Разработка концепции САПР

ГЛАВА 1

Подготовка к разработке

В *части I* нашей книги мы рассмотрим казалось бы скучные, но необходимые вопросы — общую концепцию будущей САПР. Образно говоря, мы вначале должны решить, как будем строить наш "мост" — вдоль реки или поперек. Очень часто эти важные вопросы оставляются "на потом" ("будем строить вверх, куда надо будет — туда и положим") или вообще не задаются.

Бывали и другие крайности (особенно в "советские" времена) — дальше разработки концепций, технических заданий и проектов дело не двигалось. На работу с ненужными бумагами зачастую тратились все силы и средства, а до реализации дело и не доходило.

Теперь другое время, бесполезную работу никто не делает (надо верить в лучшее, надо), но пренебрегать полезным опытом прошлого не следует. Хотя *стандарты* Единой системы программной документации (ЕСПД), введенные в СССР в начале 80-х годов прошлого века, безнадежно устарели, из них можно почерпнуть много полезного. Разумеется, с учетом нынешнего состояния технологии. Приведем перечень некоторых стандартов.

- ГОСТ 19.001-77 "Общие положения".
- ГОСТ 19.101-77 "Виды программ и программных документов".
- ГОСТ 19.102-77 "Стадии разработки".
- ГОСТ 19.104-78 "Основные надписи".
- ГОСТ 19.105-78 "Общие требования к программным документам".
- ГОСТ 19.106-78 "Требования к программным документам, выполненным печатным способом".
- ГОСТ 19.201-78 "Техническое задание, требования к содержанию и оформлению".
- ГОСТ 19.202-78 "Спецификация, требования к содержанию и оформлению".
- ГОСТ 19.401-78 "Текст программы. Требования к содержанию и оформлению".
- ГОСТ 19.402-78 "Описание программы".
- ГОСТ 19.501-78 "Формуляр. Требования к содержанию и оформлению".
- ГОСТ 19.502-78 "Общее описание. Требования к содержанию и оформлению".

- ГОСТ 24.201-79 "Требования к содержанию документа "Техническое задание".
- ГОСТ 24.202-80 "Требования к содержанию документа "Технико-экономическое обоснование создания АСУ".
- ГОСТ 24.204-80 "Требования к содержанию документа "Описание постановки задачи".
- ГОСТ 24.207-80 "Требования к содержанию документов по программному обеспечению".
- ГОСТ 34.201-89 "Виды, комплектность и обозначение документов при создании автоматизированных систем".
- ГОСТ 34.601-90 "Автоматизированные системы. Стадии создания".
- ГОСТ 34.602-89 "Техническое задание на создание автоматизированной системы".
- ГОСТ 34.603-92 "Виды испытаний автоматизированных систем".

Тексты этих документов, а также других нормативных правовых актов в области информатизации можно найти на интересном сайте **www.internet-law.ru**.

Требуется ли техническое задание

Важнейшим документом, без которого нельзя начинать работу, является *техническое задание*. Даже если вы разрабатываете программные средства по собственной инициативе, "для себя" или "ради интереса" (зарубежные читатели причин этого просто не поймут) — обязательно подготовьте техническое задание. Хотя бы для себя. Даже если вы в одном лице и заказчик, и разработчик.

Техническое задание содержит совокупность требований к программному средству и может использоваться как критерий проверки и приемки разработанной программы. Достаточно полно (и хитро) составленное (с учетом возможности внесения дополнительных разделов) и принятое заказчиком и разработчиком техническое задание очень облегчит работу над проектом.

Стандарты ЕСПД носят рекомендательный характер, но если в договоре на разработку программных средств на них будут сделаны ссылки, то для разработчиков они станут обязательными. Разработчикам при подготовке или рассмотрении договора следует быть очень внимательными, т. к. "невинная" ссылка в договоре на устаревший стандарт может потребовать много дополнительной работы (или послужить причиной для формального отклонения выполненной работы). Требований стандартов лучше придерживаться, хотя бы в структурном плане.

Раз уж мы взялись за разработку собственной системы, мы и приведем пример технического задания. Но прежде чем написать задание, необходимо продумать и задокументировать общую концепцию системы. Рассмотрением концепции системы мы и займемся в ближайших главах.

Что нужно знать об авторских правах

Может быть не в самый первый день работы, но как можно раньше решите вопрос с *авторскими правами* на будущий программный продукт. В соответствии с законо-

дательством РФ программы для электронно-вычислительных машин и базы данных, созданием которых мы намерены заняться, относятся к "произведениям, являющимся объектами авторского права". Этот факт, о котором забывают или не знают в начале работы, может оказаться очень серьезным при переходе к распространению программного продукта.

Если вы являетесь "программистом по найму", то есть работаете "на дядю" за зарплату, то вопрос с авторскими правами или решен в контракте, или будет решаться по закону. В России действуют Закон РФ № 3523-1 от 23 сентября 1992 года "О правовой охране программ для электронно-вычислительных машин и баз данных", Закон РФ № 5351-1 от 9 июля 1993 года "Об авторском праве и смежных правах" и другие правовые акты.

В этой книге мы не будем цитировать законы. Внимательно изучите их, и вы узнаете массу полезных для себя вещей. Просто удивительно, сколько российских программистов не имеют понятия о правовых актах, непосредственно регулирующих их деятельность!

В России разработка программ, особенно относящихся к САПР, часто производится не в порядке выполнения служебного задания, а по собственной инициативе авторов. Нередко разработчик вообще не является "штатным" программистом, а разрабатывает программы в дополнение к основной проектной или конструкторской работе и в рабочее, и в нерабочее время, в том числе на собственной технике. Дополнительного вознаграждения он не получает, разве что иногда начальник похлопает по плечу и скажет, к примеру, — "Молодец, Вася!"

Непременно постарайтесь "узаконить" свою программистскую деятельность или придать ей определенный статус. Если никто не собирается вам оплачивать разработку программ, то добейтесь того, чтобы в вашем контракте было упоминание о том, что разработка программ не входит в ваши служебные обязанности. А если все-таки входит, то только тех, которые включены в производственные планы и задания. Постарайтесь оговорить условие, чтобы с вами по каждой теме или на работу в целом заключался договор по какой-то из следующих существующих форм:

- ❒ авторский договор о передаче прав на использование произведений (программных продуктов), созданных в порядке выполнения служебного задания;
- ❒ авторский лицензионный договор на коммерческое использование программного средства;
- ❒ договор на создание и использование программы для ЭВМ или БД (базы данных) в порядке выполнения служебных обязанностей или по заданию работодателя;
- ❒ трудовой контракт с разработчиком программ для ЭВМ или БД при приеме на постоянную работу;
- ❒ трудовой контракт с разработчиком программ для ЭВМ или БД при приеме на работу, определенную целью задания.

Если такие договоры будут заключаться, то за работу вы будете получать оплату, а если не будут заключаться (что более вероятно), то у вас будут развязаны руки — и личные, и имущественные права на создаваемые программы будут принадлежать вам.

Если работа выполняется творческим коллективом, необходимо знать, что непременно возникнут и такие вопросы:

- выделение частей программного продукта, имеющих самостоятельное значение, их авторов и определение долей этих частей в коллективном произведении;
- определение вклада соавторов в часть произведения, составляющую неразрывное целое;
- рано или поздно выяснится, что если в числе соавторов имеется лицо, обладающее по отношению к другим соавторам властными полномочиями, то потребуется указать конкретный вклад такого лица в произведение;
- возможно, что для выполнения некоторых работ вам придется привлекать исполнителей со стороны. Непременно четко и недвусмысленно определите условия их участия в проекте в части возможных претензий на соавторство.

Как собрать команду разработчиков

Наверняка вам придется работать над проектом в составе команды. Пусть это будет очень маленькая команда (2—3 человека), но работать придется совместно. В идеале это должна быть команда друзей и единомышленников.

Сразу необходимо решить вопрос о руководителе проекта, чтобы потом, подобно известным литературным героям, не задавать друг другу вопрос: "А ты кто такой?"

Не обязательно, чтобы все в команде были профессионалами высокого класса (где их взять?), но руководить проектом непременно должен профессионал-проектировщик (напоминаем, что мы рассматриваем в книге САПР и только САПР на базе AutoCAD). Нештатными членами команды обязательно должны быть несколько конечных пользователей-энтузиастов, которым можно поручать работы, не связанные с программированием (например, формирование библиотек блоков) и тестирование готовых программ. В качестве вознаграждения за дополнительную нагрузку они должны иметь повышение производительности своего труда.

Очень важно, чтобы все члены творческого коллектива, даже если они выполняют только мелкие "черные" работы, ощущали себя участниками большого общего дела, знали его состояние, результаты и проблемы. Вспомните старую притчу, в которой на вопрос к двум рабочим, несущим носилки, о том, что они делают, один отвечает: "Я таскаю камни", а другой: "Я строю Храм". Постарайтесь создать у всех участников проекта ощущение, что они именно "строят Храм".

Необходимо периодически проводить общие совещания (даже если они не нужны технологически), специально организуя их с некоторой, возможно излишней, официальностью, а иногда и с торжественностью. Постарайтесь как можно чаще и публичней отмечать достижения конкретных людей. Можете даже придумать шутливые награды, медали и грамоты. Поверьте нашему опыту, людям это нравится (даже если они ворчат: "лучше бы материально") и такие награды могут храниться много лет.

Очень важно, чтобы молодые и малоопытные члены команды во время работы над проектом ощущали реальную пользу для себя в виде повышения собственной квалификации. Руководитель проекта просто обязан этим заниматься. Лучше всего, если он будет поручать выполнение все более сложных задач. Возможно, опытный

работник решит такую задачу за несколько минут, но в долгосрочном плане важнее обеспечение роста квалификации молодежи.

Заранее договоритесь со своим коллективом о порядке использования главного вашего достояния — исходных текстов. Возможно, что кому-то потребуется использовать ваши совместные решения в своих "левых" разработках. Совместно решите, можно ли это делать, а если можно, то как. Позаботьтесь о сохранности ваших совместных архивов. Больше всего опасайтесь доступа к ним вездесущих "приятных во всех отношениях ребят", особенно тихих и скромных. Помните: никто не ведет себя тише комара, сосущего кровь.

Задумайтесь и о возможных конфликтных ситуациях.

- Как быть, если часть коллектива перейдет на другую работу? Как придется при этом "делить" исходные тексты?
- Как быть, если кто-то перейдет в "конкурирующую фирму"? А если это будет "главный идеолог" проекта?
- Как быть, если коллектив вообще перессорится?

Мы не даем рекомендаций о том, как поступать в таких случаях. Важно, чтобы вы сами были готовы к решению этих вопросов.

Как организовать работу над проектом

Помимо согласованных в техническом задании требований к конечному продукту (что сделать) необходимо сразу же выработать и стандартные правила работы (как делать). Не обязательно, чтобы они были изложены в виде документа, хотя, разумеется, лучше закрепить их на бумаге.

Существует немало стандартов организации программирования. Например, широко известна концепция "агрессивного" и высокоэффективного процесса разработки программ Extreme Programming (сокращенно XP) — небольшой набор конкретных правил, позволяющих максимально эффективно выполнять требования современной *теории* управления программными проектами[1]. К сожалению, для тех, кто занимается разработками приложений к системе AutoCAD, особенно в России, эти правила мало пригодны, прежде всего потому, что даже теоретически нельзя разрабатывать такие серьезные проекты с привлечением столь несерьезных людских и финансовых ресурсов. Для *нашего* разработчика, который часто является и руководителем, и постановщиком, и программистом в одном лице (да еще одновременно и чертежником), концепция XP может показаться утопией, хотя отдельные элементы XP можно с успехом использовать.

Хранение информации

Непременным условием является обязанность всех соразработчиков хранить всю свою информацию на сервере (в крайнем случае, при отсутствии мощной сети, должно быть ежедневное "складирование" всех работ в специальные каталоги).

---

[1] Материалы по Extreme Programming на русском языке можно найти на сайте **www.xprogramming.ru**, оригинальное описание — на **www.xprogramming.com**.

Для каждого дня создавайте специальный архив с именем по текущей дате, например, "07_02_03". При этом никогда не удаляйте "ненужные" архивы. Поверьте, обязательно будут возникать ситуации, когда понадобится вернуться к предыдущей версии с минимальными временными потерями. Не забывайте об известном законе: "То, что вы храните достаточно долго, можно выбросить. Как только вы что-то выбросите, оно вам понадобится".

Документирование работ

В общедоступном каталоге непременно должен находиться план проекта и TODO-файлы.

Совет

Непременно записывайте все свои планы и мысли (даже не очень четкие). Проще всего это делать с помощью обычного Блокнота (notepad.exe). Не все знают, что если в первой строке файла, открываемого в notepad.exe, стоит .LOG, то "блокнотик" автоматически вставляет в текст строку со временем и датой. Ведите такие заметки в файлах с расширением .todo, "закрепите" за этим типом файлов notepad.exe. В первый раз записывайте строчку .LOG вручную, а далее вам будет помогать программа.

Листинг 1.1. Пример TODO-файла

```
.LOG
23:38 10.01.03
Начать и кончить!
23:38 10.01.03
Написать совет по использованию NOTEPAD с .LOG
```

Сразу начинайте документировать разрабатываемые программы. Ни в коем случае не оставляйте это "на потом"! В конце работы вы непременно окажетесь в цейтноте, а восстанавливать по памяти прошлогодние мысли гораздо труднее. Во многих программах не используются прекрасные возможности только потому, что разработчики о них уже забыли, а пользователи еще не знают. И могут никогда не узнать.

Лучше всего делать документацию в файле проекта справочной системы. Мы используем программный продукт Help and Manual. Эта (и другие подобные системы) позволяют вести в своем проекте структурированные справочники, из которых впоследствии можно легко сгенерировать файлы формата HLP, CHM и Manual в формате RTF. Заодно можно получить и файлы формата HTML.

Начинайте все документировать, приступая к обдумыванию концепции!

Планирование

При разработке концепции сразу составьте предварительный *план-график разработки* системы, пока не связывая его с определенными сроками. В плане необходимо установить очередность разработки компонентов системы на принципиальном уровне (детали вы пока просто не знаете). На первом этапе должна быть запланирована разработка общего ядра системы, затем — реализация задач общего характера, и только в последнюю очередь — частные задачи.

Даже если от вас требуют в первую очередь "автоматизировать электрику", убедите заказчика[1], что начинать с этого нельзя. Надо сделать так, чтобы как можно больше пользователей быстро получали эффект от вашей работы. В решении частной сложной задачи (например, полная автоматизация конструирования монолитных участков) вы можете увязнуть навсегда, и пользы для организации от этого будет немного. Если же вы быстро автоматизируете вычерчивание таблиц, общих данных, вставку блоков и других подобных работ *в общем виде*, то это сразу даст большой эффект.

Концентрируйте усилия на завершении самых важных задач, выбранных заказчиком по вашим рекомендациям, вместо того чтобы иметь несколько незаконченных задач, выбранных разработчиком.

Постоянно корректируйте и детализируйте рабочий план. В идеале он должен быть доведен до отдельных законченных задач (называемых в концепции XP *итерациями*), на решение которых отводится одна неделя. В начале разработки в итерации будут входить скрытые от заказчика внутренние задачи (например, "разработка библиотеки для работы с блоками"), а после некоторой "точки перелома" — решения для конечного пользователя (например, "вычерчивание трубопроводов в три линии"), которые будут пригодны к практическому использованию.

Замечание

Относитесь к планированию и к контролю за соблюдением графиков достаточно серьезно, но не дайте превратиться этому процессу в самоцель. Планов и графиков *у нас* всегда много, как бывает с результатами — вы прекрасно знаете (читайте В. С. Черномырдина). Скорость работы программиста не всегда возможно (в отличие от скорости рытья траншей экскаватором) рассчитать, но наличие планов и проверка их выполнения лучше, чем хаотичная работа.

Программирование

Стандартные приемы работы, связанные с непосредственной разработкой программ (кодирование, дизайн, тестирование) мы рассмотрим в *части III*.

Как строить отношения с руководителями

Вы должны быть готовы к неизбежным неприятностям с непосредственными формальными руководителями. Большинство российских руководителей "советского" или "нового русского" образца. Им, например, может нравиться "чисто конкретная" работа программиста категории "мальчик по вызову", который сделал себя просто незаменимым, потому, что без него никто не может заменить бумагу или картридж в плоттере, а иногда и просто включить компьютер. Обычно "начальникам" очень не

[1] Определитесь сразу, кто же заказчик работы. При работе "на сторону" это тот, с кем заключен договор и кто платит деньги. А если вы работаете внутри своей организации, то тут все сложнее. Официальный заказчик всегда один, а "представителей" найдется много. Командовать вами захотят многие, требования и пожелания будут противоречивые. Сразу договоритесь, кто из "заказчиков" самый главный (например, главный инженер) и будет утверждать принципиальные решения и очередность выполнения крупных разделов, а кто рангом пониже (например, начальники отделов) — с ними надо будет планировать отдельные направления.

по душе, когда целая команда занимается неизвестно чем, пишет какие-то непонятные иероглифы, переговаривается на непонятном языке и "вообще, воображают себя умнее всех".

Постарайтесь овладеть искусством дипломатии. Не забывайте старую истину: "Начальник не всегда прав, но он всегда начальник". Не давайте ему чувствовать себя дураком (свое мнение держите при себе). Постарайтесь вовлечь его в процесс работы над проектом.

- Почаще спрашивайте его мнение по каким-нибудь мелким вопросам.
- Непременно обучите его нескольким жаргонным программистским словам и аббревиатурам, чтобы он мог щегольнуть ими в своем кругу.
- Научите его правильно произносить название вашего проекта (это необходимое, но достаточное условие его участия).
- Как можно чаще давайте пощупать (именно физически пощупать, в виде твердой копии) очередные результаты ваших трудов, например чертежей, сделанных (или якобы сделанных) исключительно с помощью вашей системы.
- Периодически предоставляйте ему письменные отчеты о проделанной работе. Лучше всего сопровождать отчет перечнем новых или обновленных файлов, который вы получаете легко автоматически, а выглядит он очень солидно, и наглядно показывает, "чего и сколько" (помните фразу Аркадия Райкина?) сделано.
- Усыпив бдительность руководителя, постарайтесь юридически оформить вашу деятельность по одной из упоминавшихся ранее форм, разумеется, в выгодной вашему коллективу формулировке. Идеально, если там будет указано что-то вроде "все права на исполняемый код принадлежат работодателю, на исходные тексты — работнику" (известный принцип заключения договоров с медведем — "вершки и корешки").
- Ни в коем случае не ввязывайтесь в открытые конфликты и "нетехнические" интриги. Любой начальник сильнее вас в этих вопросах, это "его поле".
- Не пытайтесь угрожать ему своим уходом — как правило, это бесполезно. Даже если на вас держится вся работа, даже если она развалится вообще — начальнику на это наплевать.

В случае если разрыв произойдет, непременно зафиксируйте документально, что и в каком состоянии остается после вашего ухода. Скорее всего, с вас могут долго требовать какие-нибудь книги, которые вы брали несколько лет назад, вспомнят про какую-нибудь логарифмическую линейку, числящуюся за вами с незапамятных времен, но никто не вспомнит про главное богатство — *информационные ресурсы* (официальный термин по закону). Для того чтобы потом на вас не "вешали всех собак", сделайте списки всех файлов во всех каталогах, представляющих интеллектуальную ценность, командой `dir /s` (или иным способом) и приложите копию этого списка к акту сдачи-приемки.

Как нормировать разработку программных средств

В СССР нормирование труда было поставлено очень хорошо. Нормировалось, контролировалось и регламентировалось все. Разумеется, за исключением труда управ-

ленцев. Даже для проектных организаций существовали "Единые нормы времени и расценки" (ЕНВиР) на работы, выполняемые сдельно. Однако применялись эти нормы на практике очень редко по множеству причин, которые выходят за рамки нашей книги.

Предпринимались попытки нормирования и в области разработки программного обеспечения, например:

- Укрупненные нормы времени на разработку программных средств вычислительной техники, издание 1988 г.
- Укрупненные нормы времени на изготовление и сопровождение программных средств вычислительной техники, издание 1988 г.
- Типовые нормы времени на программирование задач для ЭВМ, издание 1989 г.
- Временная методика определения объемов программных средств вычислительной техники, издание 1989 г.
- ОСТ 4.071.030. Автоматизированная система управления предприятием. Создание системы. Нормативы трудоемкости.

Сейчас эти нормы выглядят просто курьезами. Главным их предназначением, на наш взгляд, было оправдание существования многочисленного "способствующего" персонала и руководства. На работу настоящих программистов они никакого влияния не оказывали.

Теперь совершенно другие социально-экономические и технические условия для программистов. Вообще-то все хотят видеть какие-то нормативы. Одни — чтобы убедиться, не переработали ли они, другие — чтобы знать, сколько же их работа стоит. Конечно, существуют рыночные отношения, но всегда полезно иметь представление о какой-то "стартовой цене" своего труда, с которой можно начинать "аукцион".

Несколько лет назад мы, на основании собственной статистики, составляли такие нормы. Нормировались работы, выполняемые с использованием системы AutoCAD. Была произведена статистическая обработка следующих наработанных данных:

- 2 382 файла общим объемом 208 979 349 байт в банке данных геоинформационной системы г. Кургана, в том числе:
  - 1 361 файл топографических планшетов и электронных карт;
  - 741 файл баз данных;
- 1 365 файлов объемом 26 065 536 байт интегрированной системы InCAD;
- 2 553 файла объемом 7 233 454 байта исходных текстов программ интегрированной системы InCAD.

Нормы были разработаны на такие виды работ:

- подготовка различных документов;
- оцифровка топографических планшетов;
- занесение информации в базы данных;
- определение геодезических координат по почтовым адресам;
- формирование банка данных из электронных калек (слоев, разложенных по отдельным файлам);

- обмен информацией между базами данных и AutoCAD;
- постановка задач для разработки программ;
- разработка программ на AutoLISP и других языках программирования;
- разработка программной документации;
- внедрение программ;
- приемка и освоение программных средств;
- подготовка программных средств к распространению;
- поставка программ;
- сопровождение программных средств в течение одного года;
- обучение пользователей;
- оказание технической помощи в эксплуатации программ;
- разработка меню в формате AutoCAD;
- создание типовых блоков в формате AutoCAD;
- формирование библиотеки из готовых блоков;
- создание слайдов и библиотек слайдов в формате AutoCAD и другие виды чертежных и расчетных работ с использованием AutoCAD.

Нормы времени были разработаны с учетом возможности автоматизированного учета и расчета трудозатрат и ориентированы, как правило, на машинно-определяемые единицы измерения объемов работ:

- 1 Кбайт размера файла текстового формата;
- 1 Кбайт размера файла формата документа текстового процессора Microsoft Word без включенных в файл шрифтов;
- 1 Кбайт размера файла рисунка AutoCAD без включения неиспользуемых объектов (блоков, описаний слоев, видов и т. п.);
- 1 Кбайт размера файла слайда AutoCAD;
- 1 Кбайт размера файла библиотеки слайдов AutoCAD;
- 1 Кбайт размера файла откомпилированного меню AutoCAD;
- 1 Кбайт размера файла откомпилированной LISP-программы для системы AutoCAD;
- 1 Кбайт размера файла базы данных формата DBF с учетом полей примечаний;
- 1 Кбайт суммы размеров файлов различных форматов, входящих в комплект программного средства или комплекса банка данных и т. п.

Объемы работ при использовании машинно-определяемых единиц измерения определялись по размеру файлов соответствующих типов. Конечно же, была разработана специальная программа, которая позволяла быстро определить трудоемкость любого вида работ и составить развернутую и сводную смету стоимости работ. Кроме того, на основные виды работ были выведены средние удельные показатели, относящиеся как к размеру файлов, так и к физическим объемам (лист, планшет).

Пример норм времени на разработку программ

Для примера приведем некоторые наши нормы времени на разработку LISP-программ (табл. 1.1).

Таблица 1.1. *Нормы времени (фрагмент) по категориям сложности, отнесенные к 1 Кбайт откомпилированного кода, чел./час*

Вид работ	Категория				
	1	2	3	4	5
Постановка задачи	9,524	11,235	12,824	22,291	28,244
Программирование	4,086	4,822	5,504	9,567	12,122
Разработка документации	6,531	7,708	8,798	15,292	19,376
Внедрение	3,018	3,562	4,065	7,066	8,953

Приведенные нормы времени применяются при определении трудозатрат по отдельным программам. Категории сложности функций определяются по табл. 1.2, а нормы времени — по табл. 1.3.

Таблица 1.2. *Категории сложности LISP-функций*

Категория	Описание
1	Вывод сообщений, запуск внешних программ без обработки результатов, работа с единицами измерений
2	Параметрическое рисование по точкам, обработка меню, вставка блоков и форм, управление слоями, обработка имен файлов и каталогов, ввод данных в диалоговом режиме, геометрические функции
3	Работа с внешними файлами, обработка видовых экранов, работа с INI-файлами, преобразование примитивов (расчленение, объединение), программное изменение свойств примитивов
4	Функции сложной обработки строк, списков, графической базы данных; запуск внешних программ с обработкой результатов; резидентные библиотеки функций; экспорт-импорт ИНФО-объектов и графики; модификация ИНФО-объектов; настройки параметров системы; генерирование меню и сценариев
5	Функции рисования сложных объектов с модификацией графической базы данных, расчеты с использованием графики, сложное многовариантное рисование

Сейчас эти нормативы несколько устарели и мы не приводим их в книге полностью ради экономии драгоценного места. "Скачать" их можно с сайта **www.gis.kurgan.ru**. Но раз наша книга о том, *как это делается*, кратко опишем применявшуюся методику определения нормативов на разработку программных средств.

Исходные тексты программ (более 2 тыс.) независимо анализировались несколькими экспертами (в том числе и авторами). Каждый эксперт, на основании собственного опыта, устанавливал для программы категорию сложности и вероятное время

Таблица 1.3. *Нормы времени (фрагмент), отнесенные к 1 Кбайт откомпилированного кода всех файлов, входящих в программный продукт, чел./час*

Вид работ	Норма времени
Подготовка к распространению	0,025
Поставка продукта	0,190
Сопровождение продукта в течение 1 года	0,010
Обучение пользователей	0,035

разработки такой программы (автор выставлял реальное время). Оценки экспертов сравнивались и усреднялись. Полученная трудоемкость относилась к машинно-измеряемым единицам этих же программ (размеру откомпилированного файла). Во время работы выяснилось, что сторонние эксперты обычно указывали большую трудоемкость, чем авторы. Такие работы, как постановка задачи, сопровождение и составление документации определялись в процентах от трудоемкости разработки. В этом случае все эксперты посчитали возможным ориентироваться на "советские" нормы.

Работы по оцифровке нормировались вообще автоматически, т. к. при работе исполнителей всегда автоматически (в тайне от операторов) формировались файлы, в которых записывалось реальное время работы. Остальное было делом техники.

Пример расчета трудозатрат

Приведем очень условный пример. Допустим, в состав нашей системы будет входить 100 функций (программа на языке LISP — тоже функция). Пусть средняя категория сложности будет 3, а средний размер откомпилированного файла — 10 Кбайт.

Общий размер файлов составит 1000 Кбайт, а предварительный расчет трудозатрат будет выглядеть так, как это показано в табл. 1.4.

Таблица 1.4. *Пример предварительного расчета трудозатрат*

Вид работ	Размер файлов, Кбайт	Норма времени	Всего, чел./час
Постановка задачи	1000	12,824	12824
Программирование	1000	5,504	5504
Разработка документации	1000	8,798	8798
Внедрение	1000	4,065	4065
Итого:			31191

Общая трудоемкость работы составит 31191/8 = 3898 чел./дней или 17,7 чел./лет. Это означает, что команда из 3-х человек должна работать 5 лет, прихватывая еще вечера и выходные дни. За это время ваш продукт будет уже никому не нужен, вы его будете многократно переделывать, вряд ли найдется работодатель, готовый

столько времени оплачивать ваш труд ради неясных перспектив. А делать это за свой счет даже при "советском" энтузиазме невозможно.

Может быть, неверные нормы? Давайте проверим. Посмотрите в каталоге Support файл xplode.lsp (в откомпилированном виде он имеет размер 8 Кбайт, и его категория сложности — примерно 3). Попробуйте представить, за какое время вы лично сможете написать такую же работающую программу.

По нашим нормам это должно быть 8 × 5,504 = 44 часа, то есть полная рабочая неделя. И это только кодирование, без учета обдумывания идеи (постановка задачи) и составления документации.

Возможно, вы думаете, что лично вы такую программу за полдня напишете? Очень может быть. Тогда вам надо немедленно бросать свою "службу", заняться исключительно разработкой программ и быстро "догнать и перегнать" Билла Гейтса.

Но, скорее всего, разработка действительно работающей программы, на которую вы планируете затратить один день, займет вот сколько времени (по известному шутливому, но точному алгоритму определения сроков):

1. Дни переводим в единицу измерения следующего порядка (недели).
2. Умножаем планируемый срок на 2 (а лучше на 3).
3. Добавляем еще 3 дня, которых всегда не хватает.

Итого получится две недели и три дня. И это еще оптимистичный прогноз!

Однако вернемся к нашей смете. Выбрасывать ее мы не будем. Пусть она останется для демонстрации начальнику, о котором мы упоминали. Но нам необходимо получить реальные трудозатраты (хотя бы для себя). Поэтому по российской привычке мы начинаем "кроить" смету.

Возьмемся за сомнительные позиции. Само программирование составляет только 17% от общего объема работ, его пока трогать не будем. Первым кандидатом на "урезание" является строка "Постановка задачи". В состав этих работ по нашим нормам (и традиционно) входят:

> *"Сбор исходных данных. Выбор и обоснование критериев программных средств. Определение структуры входных и выходных данных. Выбор методов работы. Определение требований к техническим средствам. Определение требований к задаче. Определение стадий, этапов и сроков разработки программных средств. Выбор языков программирования. Отладка и сдача в опытную эксплуатацию. Проведение всех видов испытаний. Совместно с программистом: корректировка программ и документации по результатам испытаний".*

Это формальная формулировка, фактически это затраты на обдумывание идей, споры (возможно, с самим собой) и пр. При разработке *отдельной*, самостоятельной программы эти затраты действительно превышают затраты на "кодирование" примерно в два раза. Но при разработке множества программ, входящих в состав большой системы, такие накладные расходы значительно меньше. По нашему мнению, для системы в целом их можно принять около 5% от затрат на постановку отдельной программы (просто уменьшив размер файлов при расчете, чтобы не ломать алгоритм), то есть в нашем случае это составит 641 час.

Вторым кандидатом на "секвестр" является разработка документации. Выбросить ее нельзя, но затраты на написание одного тома по 100 программам значительно

меньше, чем для 100 томов по каждой программе. Отведем на разработку документации 10% от первоначально планируемого объема, имея в виду еще и экономию на лаконичном изложении. Итак, затраты на документацию составят 880 часов.

Теперь возьмемся за статью "Внедрение". В состав работ по внедрению входят:

> *"Подготовка и передача программ и документации для сопровождения и (или) изготовления. Проверка алгоритмов и программ. Опытная эксплуатация задач. Подготовка документации и сдача в промышленную эксплуатацию".*

Эту статью затрат мы можем смело выбросить. Не потому, что это не нужно, а потому, что это можно совместить по срокам с другими работами и переложить на будущих пользователей. Эти работы в современном понимании являются "Бета-тестированием" и всегда лучше, если тестирование будут выполнять сами пользователи. Некоторые фирмы (не будем показывать пальцем) за право бета-тестирования еще и деньги берут! Но мы, по скромности, делать этого не будем. Да и кто нам даст.

Продолжаем приятную процедуру урезания сметы. У нас осталась статья "Программирование". Причем только сам процесс перевода на язык программирования готовых алгоритмов. Можно ли здесь экономить? Можно и нужно.

Достигается это путем сокращения объема кода, который будет писать программист за счет повторного использования типовых функций, компонентов, классов, интерфейсов. Как это делается, мы и будем разбирать в основной части книги. Идеальным результатом является такой, когда конкретная программа, делающая очень много с точки зрения конечного пользователя, программистом пишется в одну-две строчки, представляющих вызов готовых функций. Технику и технологию такого программирования мы будем рассматривать позже. Пока же, по нашему опыту, мы сократим затраты на программирование в 10 раз.

Итак, наш новый расчет трудозатрат будет выглядеть так, как это показано в табл. 1.5.

Таблица 1.5. *Предварительный расчет трудозатрат*

Вид работ	Размер файлов, Кбайт	Норма времени	Всего, чел./час
Постановка задачи	50	12,824	641,2
Программирование	100	5,504	550,4
Разработка документации	100	8,798	879,8
Итого:			2 071,4

Теперь трудоемкость работы составит 2 071,4/8 = 258,9 чел./дней, что вполне реально. Остается эти планы реализовать (что уже менее реально). Кроме того, здесь мы не учли множество вспомогательных работ (подготовку блоков, типов линий, штриховок, слайдов, меню), а также разработку программ на других языках программирования.

В конце книги мы постараемся пронормировать выполненную нами работу по разработке нашей САПР.

Как определить стоимость работы

Имея расчет трудозатрат, можно более или менее точно определить стоимость работы.

Зная *трудозатраты исполнителей* и *среднюю часовую зарплату исполнителей*, можно получить *общую заработную плату исполнителей*. Часовая зарплата исполнителей устанавливается на предприятии в зависимости от множества факторов и в разных регионах может отличаться на порядок.

Полученная сумма зарплаты умножается на некий коэффициент, учитывающий отчисления на различные налоги, исчисляемые от фонда зарплаты. Величина этого коэффициента меняется в соответствии с текущим законодательством (в том числе и местным). Полученная величина называется *начислениями на зарплату*.

Далее определяются *накладные расходы* предприятия, составляющие 300—500% от суммы заработной платы. Накладные расходы учитывают все другие затраты предприятия, кроме основной зарплаты (зарплата вспомогательного персонала и руководства, аренда помещений, платежи за электроэнергию и пр.).

По сумме основной зарплаты, начислений на зарплату и накладных расходов определяется *себестоимость* продукции. *Цена реализации* образуется путем добавления к себестоимости продукции плановых накоплений, величина которых колеблется в зависимости от "аппетитов" предприятий в пределах 10—50% (в "советские" времена норма плановых накоплений была 6%). После добавления к цене реализации налога на добавленную стоимость получается *отпускная цена* продукции.

Приведенная схема ценообразования соответствует типичной затратной методике. В данном случае мы рассматривали "расчет по трудозатратам", но существуют и более сложные методы составления смет затрат, учитывающих стоимость материалов, эксплуатацию механизмов и другие затраты. В любом случае при "затратном" ценообразовании производитель продукции стремится возместить все свои реальные или вымышленные затраты за счет потребителя. Предприятиям-монополистам это с успехом удается. Краткое описание такой методики мы привели только потому, что очень много разработчиков программных средств вообще не имеют представления об этих вопросах.

В рыночных условиях цена продукта определяется совсем другими факторами. По приведенной методике вы можете только определить, сколько вы "хотели бы съесть". Но сколько вам "даст съесть" потребитель вашей продукции — совсем другой вопрос, к которому мы вернемся в самом конце книги.

ГЛАВА 2

Анализ и постановка задачи

В этой главе мы займемся общей постановкой задачи, т. е. постараемся сформулировать, *что* мы должны сделать.

Приступая к разработке системы автоматизированного проектирования, необходимо "оглядеться", разобраться, какие аналоги имеются на рынке, какая система нужна нам, чем она должна отличаться от других.

Какие САПР существуют

Официальной классификации САПР в России не существует. Не существует даже официального толкования термина "система автоматизированного проектирования". Неофициальные определения выглядят примерно так: "комплексная программно-техническая система, предназначенная для выполнения проектных работ с применением математических методов".

Кто-то считает, что компьютер с программным обеспечением для проектирования это уже САПР, для кого-то САПР — система, не иначе как объединяющая в себе проектирование, разработку документации, ее выпуск, хранение, изменение и управление проектами.

Любая САПР является неразрывным комплексом, объединяющим пользователей, программное обеспечение и технические средства. В нашей книге мы будем только косвенно касаться технических средств, а под термином "САПР" будем, в контексте книги, понимать только программные средства для автоматизации проектирования. Книга посвящена разработке САПР на базе AutoCAD, поэтому мы не будем касаться иных систем, а рассмотрим только приложения к AutoCAD.

Приложений "под AutoCAD" существует очень много. Это и утилиты общего назначения, облегчающие работу с AutoCAD, и "настоящие" САПР, позволяющие моделировать сложные трехмерные объекты с анализом их характеристик и оптимизацией проектных решений, и системы, ориентированные на подготовку и выпуск документации, включая чертежи, спецификации, ведомости и текстовые документы.

Сама фирма Autodesk и ее партнеры пытаются "перетянуть одеяло" в сторону трехмерного моделирования и оптимизации конструкций, уделяя меньше внимание системам подготовки и выпуска "обычной" документации, которая до сих пор в основном и нужна заказчикам. Отчасти, это происходит потому, что системы высокого класса дороже — цена базовой системы AutoCAD за последние 10 лет выросла в

2 раза. Разумеется, значительно выросли и возможности системы, но далеко не все они нужны основной массе проектировщиков. Плюс к тому, ведущие идеологи САПР (они не потому ведущие, что всегда "шли за пользователем" — "чего изволите?") стремятся предвосхитить развитие потребностей своих, а желательно и "чужих", потребителей и предлагают решения, которые, по их мнению, должны дать кардинальный эффект. Ну и не надо забывать о принципиально разных идеологиях проектирования на территории экс-СССР и того мира, для которого, собственно и разрабатываются все крупные САПР. Хуже всего, что в рамках системы AutoCAD нет возможности выбора. Относительно недорогой AutoCAD LT не имеет никаких легальных средств для разработки приложений, а они очень нужны для российских условий. В результате даже для простой двухмерной графики пользователи вынуждены использовать дорогую и излишне "навороченную" полную систему AutoCAD.

Попробуйте сами, уважаемый читатель, оценить действительно необходимое количество рабочих мест с "моделирующими" САПР на вашем предприятии. Очень хорошо, если их окажется более 10%. Попробуйте также оценить работы, выполненные за прошедший год. Сколько оплачено заказчиками трехмерных моделей, сколько таких моделей и изготовленных с их помощью красивых фотореалистичных картинок помогли привлечь дополнительных заказчиков, партнеров, "выбить" участок для строительства, "облапошить" заказчика, извините, "укрепить заказчика в мысли, что он был прав, выбирая именно вашу проектную фирму"?

Безусловно, будущее за трехмерными моделирующими системами. Но только в том случае, когда все разделы проекта будут выполняться в единой системе (или в родственных системах на единой основе), а из трехмерной модели, условно говоря "нажатием кнопки", будут формироваться все требуемые чертежи в соответствии с принятыми в стране стандартами. При этом соотношение стоимости систем и проектных работ должно быть таким, чтобы проектная фирма могла приобретать САПР на доходы от продажи проектной продукции с реальной перспективой окупаемости. Скорее всего, в наступившем веке это произойдет. А до достижения "светлого будущего" проектировщикам предлагается использовать то, что имеется.

Изучая статьи в журналах, специализирующихся на САПР, вы увидите множество прекрасных иллюстраций и узнаете, что именно, и у кого именно (в зависимости от ангажированности издания) вам "следует немедленно приобрести". Сразу скажем — рекламируются превосходные продукты! Например, Mechanical Desktop, Autodesk Architectural Destktop, AutoCAD Land Development Desktop, Autodesk Civil Design, Autodesk Survey, PLANT-4D, продукты российских разработчиков — MechSoft-PROFI Plus, GEO + CAD, МАЭСТРО, ElectriCS, HydrauliCS, MechaniCS, TechnologiCS, GeomatiCS, СПДС GraphiCS и др.

Прочитав описания большинства продуктов, так и хочется бросить все и бежать к ближайшему дилеру. Сдерживает только знакомство с прайс-листами. Изучив прайслист, специалисты обычно произносят классическую фразу незабвенного Ипполита Матвеевича Воробьянинова: "Однако!". Простой расчет показывает, что для оснащения 100 рабочих мест (средняя по нынешним временам проектная организация) только минимально необходимыми программными продуктами и только "под AutoCAD" потребуется не менее $1 700 000, т. е. около $17 000 на одно рабочее место. Это при условии всех скидок на сетевые лицензии и количество лицензий, других расчетных программ и без стоимости оборудования. Именно поэтому "кое-где у нас порой" приобретают продукты у "неофициальных партнеров" Autodesk.

Однако предположим, что ваше проектное предприятие имеет большой пакет заказов, не стеснено в средствах и может приобрести все необходимые программные

продукты. Вы думаете, что "светлое будущее" наступит сразу после этого? Как бы не так!

Радуясь выгодному приобретению, вы непременно установите, что в этих замечательных системах есть все, кроме того, что вам нужно "здесь и сейчас". Отличные возможности для перспективного развития вашей фирмы есть, а всякой "мелочевки", которая нужна для выполнения текущей работы, не хватает. Конечно, все это будет, но... позже "с течением времени", "в следующей версии" (которую вы сможете приобрести со скидкой).

Вот тут вы и поймете, почему сотни российских конструкторов и проектировщиков занимаются не свойственным им делом — адаптацией системы AutoCAD и разработкой, пусть на любительском уровне, собственных программ. И созреете для дальнейшего прочтения нашей книги.

Какая САПР нам нужна

Итак, нам все-таки надо делать *свою систему*. Если надо — значит надо! Давайте сформулируем основные принципы разрабатываемой системы.

1. Это будет САПР *строительного профиля* в широком понимании, т. е. предназначенная для автоматизации строительного проектирования, реконструкции и технического перевооружения любых зданий и сооружений. Если вы "машиностроитель", то не расстраивайтесь — многое из того, что мы сделаем, пригодится и вам.
2. Мы будем делать систему низшего уровня (класса "чертилка"). О том, что моделирующие системы нужны не всем, мы только что говорили. А вот "электронных кульманов" еще долго не будет хватать всем желающим. Это означает, что с помощью программ *нашей системы* пользователь сможет начертить все, что ему нужно, но не сможет "проектировать" в высоком смысле этого слова. Например, пользователь-электрик сможет начертить любые планы и схемы электрооборудования с использованием соответствующих обозначений элементов и проводников, но не сможет по принципиальной схеме автоматически получить схему соединений, оценить протяженность проводов, выполнить автоматизированную раскладку кабелей по трассам. Это уже задачи, решаемые САПР более высокого уровня, например, ElectricCS.
3. Мы будем разрабатывать систему для двухмерного черчения. О трехмерных "десктопах" всегда позаботится фирма Autodesk и ее партнеры, и сделают они это наверняка лучше нас с вами. А пока гиганты программной индустрии разворачиваются, мы постараемся обеспечить пользователей пусть "примитивными", но эффективными средствами для выпуска документации.

Какие разделы проектов мы будем автоматизировать

Строительное проектирование включает в себя множество разделов (марок) проектной документации:

- ❒ Генеральный план.
- ❒ Транспорт.

- Архитектурные решения.
- Различные виды конструкций (железобетонные, каменные, металлические, деревянные).
- Различные виды технологии производства (от простейшей расстановки мебели до сложных технологических комплексов), включая множество видов технологических трубопроводов (сжатый воздух, мазут, смазочно-охлаждающие жидкости, специальные газы).
- Отопление, вентиляция и кондиционирование воздуха.
- Водопровод и канализация.
- Электрическое освещение.
- Силовое электрооборудование.
- Слаботочные устройства.
- Автоматизация технологических процессов и санитарно-технических систем.
- Различные виды наружных инженерных сетей (тепловые сети, водоснабжение, канализация, наружные газопроводы, технологические трубопроводы).

Рабочие чертежи всех этих разделов разрабатываются в соответствии со стандартами Системы проектной документации для строительства (СПДС). Не охвачены СПДС, но широко используются в строительной отрасли обмерные чертежи, топографические планы, чертежи по организации строительства и исполнительные чертежи.

Кроме того, во многих разделах проекта предусмотрена необходимость разработки чертежей общих видов нетиповых конструкций и нестандартизированного оборудования.

Имеющиеся на рынке САПР программные продукты обычно охватывают один или несколько перечисленных разделов. Мы поставим перед собой задачу-максимум — дать пользователям средства для "рисования" *всех строительных разделов* в рамках одной системы. Вам эта задача кажется слишком сложной? Да, это сделать непросто, но вполне реально. Далее мы это докажем практически.

Кто будет пользователем нашей системы

Создание системы, объединяющей все разделы проекта, позволит значительно расширить круг пользователей. Пользователями *такой системы* могут быть большие и мелкие проектные организации, предприниматели, проектные отделы, группы и бюро на множестве заводов, отделы и комитеты архитектуры и градостроительства, эксплуатационные организации коммунального хозяйства и другие службы. А если мы дополним *нашу систему* возможностями использования геоинформационных технологий, круг пользователей расширится еще больше. Конечно, *такая система* может применяться и студентами всех перечисленных специальностей.

Какие версии Windows и AutoCAD мы будем использовать

Этот вопрос не столь праздный, как может показаться. Вы еще не забыли, что мы должны написать техническое задание? А в техническом задании это непременно

нужно отразить хотя бы потому, что обязательно найдутся знатоки, считающие, что САПР надо делать вообще не для компьютеров с процессорами Intel ("рабочие станции на RISC-процессорах лучше!"), и не "под Windows", и т. д., и т. п.

Поэтому зафиксируем документально в техническом задании, что *наша система* будет использоваться в операционных системах Windows 98, ME, 2000 и XP. Следует отметить, что Windows 95 мы уже не принимаем во внимание, это окажется важным при решении некоторых вопросов.

Версию базового программного продукта AutoCAD мы принимаем не ниже 2002 (хотя в силу совместимости то, что будет сделано, сможет работать и в AutoCAD 2000, и в AutoCAD 2000i), с возможностью перехода на версию 2004. Это также очень важно. Пользователей AutoCAD R14, к сожалению, проигнорируем. Слишком во многом мы будем ограничены, если постараемся учесть и этот, хоть еще популярный, но снятый с производства продукт.

Как узнать, что требуется пользователям для работы с AutoCAD

Профессиональные программисты, разрабатывая САПР, часто допускают ошибку, заключающуюся в том, что они начинают делать то, что умеют, а не то, что надо, и как умеют, а не как нужно. Вторая ошибка заключается в том, что разработчик программ для одного раздела не интересуется тем, что и как делают "смежники". В результате могут, например, создаваться разные и несовместимые между собой программы составления спецификаций или вычерчивания таблиц.

Анализ проектов

Давайте посмотрим, что действительно требуется пользователям. Для этого лучше всего внимательно изучить все разделы нескольких готовых проектов. Кроме того, необходимо просто наизусть знать стандарты СПДС. Непременным условием является участие в этой работе постановщика задач с достаточно широким кругозором и не понаслышке знакомого с технологией проектирования.

Анализируя проектную документацию и прикидывая при этом, как такие результаты можно получить программным путем, мы обнаружим массу возможностей для унификации работы.

Примечание

Кстати, вы не забыли, что мы договаривались фиксировать все свои действия и планы в TODO-файлах? Непременно записывайте результаты исследований и свои наметки на пути реализации.

Общие данные по рабочим чертежам

Прежде всего, мы увидим, что для всех разделов нужны форматы с основными надписями ("штампами") нескольких видов. Форматы могут иметь различные размеры. Особого внимания требует заполнение граф основных надписей. Во всем проекте должно использоваться единое базовое обозначение, название стройки, одинако-

вые названия зданий и сооружений, название организации. Очевидно, что вычерчивание форматов и заполнение граф нужно автоматизировать с использованием единых данных. Бывалые проектировщики подскажут, что иногда, в последний момент, может потребоваться изменение названия стройки или еще чего-то. Лучше предусмотреть возможность автоматизации и этих действий.

Для всех разделов по единому стандарту выполняются общие данные (заглавный лист). Иногда общие данные располагаются на нескольких листах. Во всех разделах проекта в составе общих данных вычерчиваются по единым формам таблицы (ведомость чертежей основного комплекта и т. п.). На заполнение некоторых таблиц расходуется очень много времени. Например, ведомость ссылочных и прилагаемых документов может включать несколько десятков серий, номер и название каждой должны быть записаны "как положено". Каждый раздел проекта может иметь и специфичные общие данные (таблицы, схемы, указания), но выполнение всех их можно автоматизировать унифицированными программами. В общих данных присутствуют многочисленные текстовые "общие указания", которые явно можно писать, используя типовые заготовки.

Планы зданий

Вполне очевидным является то, что все планы должны выполняться на единой строительной подоснове. Однако строительная подоснова, используемая смежниками, значительно отличается от планов, имеющихся в архитектурно-строительной части. На них не должно быть множества строительных надписей, размеров, обозначений. Не показываются полотна дверей, штриховки, строительные конструкции выполняются тонкими линиями, показывается тонкими линиями технологическое оборудование, к которому подводятся коммуникации. Планы смежных частей могут выполняться в ином, нежели строительная часть масштабе, могут выполняться фрагменты с чертежами установок. Планы отдельных зданий или фрагментов зданий, по которым смежники являются технологами (насосные, котельные, щитовые), могут выполняться смежниками до разработки строительной части. Кроме того, в смежных разделах могут выполняться разрезы и узлы, вообще отсутствующие в строительной части, например для уточнения положения коммуникаций в межферменном пространстве, каналах и подвесных потолках.

После анализа планов становится ясно, что работу с ними необходимо облегчить, особенно за счет рационального распределения объектов по слоям в строительной части.

Трубопроводы и проводники

Особое внимание нужно уделить вычерчиванию на планах трубопроводов, воздуховодов и проводников. Трубопроводы могут вычерчиваться "в три линии" (т. е. с указанием осевой линии, реального диаметра и размера стороны, более или менее реалистичным изображением фасонных частей и арматуры) и "в одну линию". Трубопроводы, расположенные один под другим при изображении в одну линию, на планах показываются условно отнесенными от стен и расположенными параллельно друг другу. Электрические провода и кабели на планах вообще показываются довольно условно, главным образом для отображения направления трасс. Общий принцип заключается в том, что строительный план коммуникаций внутри здания — это совсем не то, что вид в плане на трехмерную модель. И это надо учитывать.

Схемы систем

Значительный объем в составе проектной документации занимают схемы различных систем. Схемы бывают "плоские" (например, технологических трубопроводов, котельных, тепловых пунктов, систем электроснабжения и автоматизации) и "аксонометрические" (системы отопления, вентиляции, водопровода, канализации, газоснабжения). Аксонометрические схемы выполняются в проекции, приблизительно соответствующей *"фронтальной изометрии с левой системой осей и коэффициентом искажения вдоль осей, условно принятым за единицу"*. Следует заметить, что это совсем не та "изометрия", которая имеется в системе AutoCAD, и получить точно такую проекцию с помощью средств AutoCAD просто не удастся.

Примечание

По нашему мнению, традиционная проекция очень неудачна, т. к. приводит к многочисленным наложениям изображений участков, с которыми борются намеренным искажением размеров. Ее единственным достоинством является компактность (длинные участки обычно располагаются параллельно стороне листа). Но попытки внедрять изометрическую проекцию, принятую в системе AutoCAD, окончились неудачно. Чересчур уж эта проекция непривычна взгляду "советских" сантехников и слишком нерационально (и это более важно) используется площадь чертежа, особенно в протяженных системах.

Аксонометрические схемы, теоретически, должны вычерчиваться в масштабе (обычно в таком же, как и план трубопроводов). Практически же масштаб схем получается весьма условным, и измерять протяженность трубопроводов по схемам нельзя. Это объясняется тем, что:

- условные обозначения элементов трубопроводов и воздуховодов, размеры которых нормируются на бумаге, не соответствуют реальным размерам;
- при наложении элементов схемы один на другой и при длинных участках, непроизводительно занимающих площадь чертежа, производятся различные вырезки и выноски участков схем.

Вычерчивание схем, при всем их разнообразии, легко автоматизируется. Все они вычерчиваются в виде линий (сплошных или с "врезанным" буквенно-цифровым обозначением), на схемах показывается арматура и другие элементы систем, проставляются отметки, диаметры, марки оборудования. В схемах, разрабатываемых в совершенно разных разделах проекта, часто используются одни и те же обозначения (оборудование, арматура, контрольно-измерительные приборы). В некоторых разделах проекта широко используются типовые укрупненные схемы, незначительно отличающиеся в разных проектах и системах (обвязки воздухонагревателей, схемы управления электродвигателями, схемы автоматизации).

Рабочее и монтажное проектирование

У читателей, мало знакомых со строительным производством, несомненно, возникнут вопросы о том, как же строятся здания, если на планах трубопроводы показываются условно отнесенными от стен, схемы изображаются "условно в масштабе", на чертежах многое не показывается, но подразумевается? Более того, в стандарте на оставлении спецификаций оборудования, изделий и материалов (которых мы пока не касались) прямо указано, что "элементы трубопроводов (отводы, переходы, тройники, крестовины, фланцы, болты, гайки, прокладки) в спецификацию не включают"! А как же тогда строить?

Попробуйте (лучше теоретически) заказать в проектной организации "советского" образца проект реконструкции вашего личного санузла с заменой сантехприборов и установкой счетчиков на холодном и горячем водоснабжении (реальная работа, которая в массовом порядке выполняется в каждом городе). Получив рабочие чертежи, полностью соответствующие стандартам СПДС, вы сначала, возможно, не увидите подвоха. Но, начав "комплектацию", т. е. закупку в магазинах нужных материалов и изделий, вы обнаружите, что в спецификации оборудования, по которой вы наивно начали закупки, имеется явно завышенное количество труб, но нет ни одной фасонной части, которые при современном монтаже из металлополимерных труб стоят очень недешево. То же касается всяких сгонов, футорок, контргаек, креплений и прочих "хитрых" деталей, о которых вы не имеете представления и не знали, что они понадобятся. При монтаже канализации выяснится, что унитаз никак не может быть установлен в проектное положение, потому что его выпуск не стыкуется с существующей чугунной трубой — требуется специальная таинственная "гофра", о которой, в лучшем случае, скажет монтажник (в худшем случае он расколет выпуск унитаза). Отправившись в магазин за гофрой, вы обнаружите десяток их разновидностей и, конечно, купите не такую, какая нужна. И это с простыми изделиями, что уж говорить про счетчики...

Если вы попытаетесь предъявить претензии проектировщикам, то вам вежливо покажут соответствующие ГОСТы, в том числе совсем свежие российские, и докажут, что в проекте все сделано "как положено". В чем же дело?

А дело в том, что по рабочим чертежам (по крайней мере, вентиляционных систем и трубопроводов) на самом деле не должен производиться монтаж систем. В течение многих лет в СССР существовала система разделения рабочего и монтажного проектирования. На основании *рабочих чертежей*, выполненных проектной организацией по стандартам СПДС за счет средств на проектные работы, монтажные организации разрабатывали (за счет средств на накладные расходы) *монтажные проекты*, включающие деталировочные чертежи трубопроводов и воздуховодов, а также деталировочные чертежи некоторых конструкций и оборудования (существовала чрезвычайно запутанная система распределения поставок оборудования и материалов между заказчиками и подрядчиками). Монтажным проектированием занимались специальные группы (а иногда и целые институты) при монтажных организациях. Правда, было это не всегда, для мелких объектов "отсталые" монтажные организации часто выполняли монтаж "по месту" и "по соображению".

При "деталировке" все участки трубопроводов и воздуховодов разбивались на узлы и детали с точным определением размеров. При этом учитывались все тонкости технологии (монтажное положение, крепления, радиусы гнутья труб и пр.). Детали и узлы изготавливались на специальных заводах и в центральных заготовительных мастерских. При этом могли изготавливаться и испытываться в заводских условиях укрупненные узлы, блоки оборудования и трубопроводов, а иногда и целые полносборные здания со всей внутренней "начинкой" (обычно для строительства на Севере).

Теперь вы поняли, какой важный этап был упущен в вашем личном проекте?

В последние годы существования СССР, по мере компьютеризации проектирования, предпринимались попытки совмещения рабочего и монтажного проектирования *(рабоче-монтажные проекты)*. Все тогда закончилось отдельными успешными экспериментами, т. к. обычные проектные организации не имели в то время соответст-

вующего технического и программного оснащения, да и финансировались эти работы из разных источников.

В современных условиях, когда наконец-то начался переход на строительство "под ключ", стали образовываться проектно-строительные предприятия (увы, их пока очень мало), полностью берущие на себя и проектирование и строительство, идея рабоче-монтажного проектирования наверняка реализуется. Большую роль в этом сыграют и трехмерные моделирующие системы, такие как Plant-4D. Но это еще впереди. А пока, даже умея смоделировать здание и трубопроводы, проектировщики часто сталкиваются с тем, что их замечательные проекты запросто могут быть забракованы органами государственной экспертизы. Люди там часто работают пожилые, с новыми технологиями они мало (мягко говоря) знакомы. Почему бы и не "завернуть" проект, выполненный в непривычном виде на формальном основании несоответствия ГОСТ? А вы потом доказывайте, что "хотели, как лучше". Очень может быть, что докажете.

Это лирическое отступление от темы программирования мы сделали для того, чтобы показать, с какими нюансами может столкнуться разработчик САПР и как много ему нужно знать помимо самого программирования.

Практическим же выводом должна быть мысль о том, чтобы сделать *нашу систему* пригодной не только для разработки рабочих чертежей по СПДС, но и для разработки монтажных деталировочных чертежей. Это сделать совсем просто. Необходимо только, чтобы все элементы систем вычерчивались не "на глазок", а точно по размерам, что программным путем сделать совсем не трудно.

Генеральные планы и наружные сети

Продолжая анализ проектной документации, рассмотрим чертежи генерального плана, транспорта и инженерных сетей всех видов. Общим у всех этих разделов является то, что все чертежи выполняются на единой топографической основе.

Топографические планы

Оригинальная топографическая основа находится и непрерывно обновляется в соответствующих городских службах. Обычно это так называемые топографические планшеты, каждый из которых изображает план города в квадрате со строго определенными координатами углов в городской системе координат. Например, для масштаба 1:500 планшеты имеют размер рабочего поля 500×500 мм и на них показаны участки местности размером 250×250 м.

В каждом городе существует местная система координат и определенная "разграфка", т. е. система нумерации планшетов. Зная разграфку и номер планшета, можно рассчитать координаты его углов и, наоборот, зная координаты точки, определить номер планшета.

В некоторых городах и на некоторых предприятиях поддерживаются электронные топографические планшеты, обычно в составе геоинформационных систем. Довольно часто такие планшеты могут использоваться в качестве подосновы для проектирования, а проектируемые объекты, в свою очередь, накладываться на дежурный электронный план города. В этом вопросе имеется масса организационных, технических и юридических тонкостей.

Для разработки проектной документации проектные организации через заказчика (инвестора) получают в работу не топографические планшеты, а специально подготовленные топографические планы, "сшитые" из нескольких планшетов, на участок строительства и трассы инженерных сетей[1].

Для автоматизированного проектирования обязательно потребуется электронная топографическая основа. Чаще всего ее придется создавать самостоятельно. Тема создания векторной электронной топоосновы заслуживает отдельной большой книги, здесь мы ее рассматривать не будем. Мы просто наметим в своих планах, что нам потребуются программные средства, позволяющие нарисовать топографический план с использованием стандартных условных обозначений. Для рисования топографии требуется:

- самое главное и самое сложное — каким-то образом получать координаты точек объектов плана (углов зданий, поворотов трасс и т. п.);
- по полученным координатам быстро отрисовывать требуемые изображения.

Ставим себе такую задачу и заносим в план.

Условные обозначения в топографии и на генпланах

Условные обозначения и изображения существующих объектов (в топографии) и проектируемых (на генеральных планах) весьма сходны, хотя и регламентируются различными документами. Анализируя их, мы можем наметить и группы программных реализаций. Все многообразие условных знаков (около 500) не так уж сложно изобразить небольшим набором программ.

Некоторые программы, значительно облегчающие работу генпланистов, сделать очень легко. Примерами являются "разбивочные" чертежи, а также определение и подписывание координат различных точек (углов участков и трасс). Такие работы очень ответственны, кропотливы, длительны, при ручном проектировании их выполняют квалифицированные специалисты графоаналитическими методами. С использованием несложных программ эти работы выполняются мгновенно.

В то же время для разделов генплана требуются чертежи, весьма нелегкие для автоматизации при двухмерном проектировании, например вертикальная планировка с планом земляных масс и расчетом баланса срезки и подсыпки. Программы для таких работ лучше отложить на вторую очередь, не пытаясь разработать их "с наскока".

Планы коммуникаций

Все планы коммуникаций также выполняются на топографической основе, с учетом проектируемых объектов и решений по вертикальной планировке. И все многообразие изображений коммуникаций также легко автоматизируется небольшим количеством программ. Конечно, для каждого раздела имеется своя специфика, но общего очень много.

---

[1] В современных условиях часто вообще ничего не получают, а работают на том, что найдут в архивах. Но это совершенно другая тема.

Профили

Практически для всех сетей разрабатываются продольные профили трасс. Это также сложная для автоматизации работа. Профиль "как в учебнике", на свободном участке, нетрудно нарисовать и программным путем. В реальных городских условиях при множестве пересечений все гораздо сложнее. Часто не бывает однозначных решений, "машине" тут все доверить нельзя. В то же время черновую работу, например вычерчивание заготовок профилей, можно и нужно делать программно. Намечаем, что полное автоматизированное вычерчивание профилей мы отложим на вторую очередь.

Схемы, узлы, разрезы

Для многих видов сетей разрабатываются схемы, узлы трубопроводов в колодцах и камерах, поперечные разрезы трассы. Все эти чертежи практически не отличаются от аналогичных для внутреннего оборудования зданий. Надо только учесть, что возможно рисование этих изображений в других единицах рисунка. Следовательно, программы, рисующие, например стены, должны понимать, какие стены (здания, теплофикационной камеры или подпорной стенки) изображаются и должны правильно работать.

Изучение специфики работы в AutoCAD

После планирования того, *что* должна рисовать *наша система*, надо наметить, *как* она должна это делать. Для этого необходимо:

- очень детально проанализировать имеющиеся рисунки, сделанные в AutoCAD, в части того, какими примитивами все нарисовано, на каких слоях, как использовались блоки, как все выглядит на экране и на бумаге;
- внимательно разобраться, как работают пользователи, попутно изучая их навыки и даже психологию (в части, относящейся к работе).

Очень полезно изучить рисунки и приемы работы в других организациях. Тут надо действовать деликатно, чтобы вас не приняли за шпиона.

Наверняка изучение чужих рисунков вызовет у вас массу разнообразных эмоций. Они бывают настолько сильны, что по этому поводу от избытка чувств, как правило, смешанного с изрядной долей снобизма, организуются специальные ветви в форумах. Вам, как разработчику, надо не поддаться эмоциям, а подумать, как сделать программы, чтобы помочь пользователю максимально избежать ошибок.

Приемы работы лучше всего изучать на практике. Если вы хорошо знаете, как рисовать строительную часть, попробуйте нарисовать (по готовому чертежу или даже по иллюстрации к ГОСТ) схему отопления. Или наоборот. Присмотритесь, как это делают другие. Тут важны мельчайшие детали. Вы можете обнаружить, что план лестничного марша, возможно, придется рисовать не только с правого нижнего угла (как вы думали), а с любого. Стены, оказывается, удобнее рисовать не двойной линией, а подобием от координационных осей (но хорошо бы, чтобы они попадали на нужный слой) с последующими сопряжениями, и т. д.

Такую практику нельзя ограничить одним днем или месяцем. Этим придется заниматься постоянно, во время всего "жизненного цикла" программ, переделывая то, что казалось вам удобным, но почему-то не пользуется симпатией пользователей.

Классификация пользователей

Любая программа должна быть ориентирована на определенный круг пользователей по квалификации и по их потребностям. Хорошо, когда программа разработана и с учетом психологии пользователей. Попробуем составить абсолютно ненаучную классификацию потенциальных пользователей и подумаем, как эти знания применить при разработках.

Пользователи системы AutoCAD делятся на классы, типы, виды в зависимости от множества факторов, например от "происхождения" ("из архитекторов", "из конструкторов" и т. п.). Тем не менее, устойчиво существуют несколько больших групп, давно получивших в интернет-сообществе имена собственные.

"Чайники"

Это начинающие пользователи, многого пока не знающие. "Чайника" можно сравнить с ребенком, который только учится ходить, но потом, возможно, будет бегать быстрее всех в мире. Вообще-то все мы "чайники", только в разных вопросах. "Чайников" надо любить, уважать и, при разработке программ, постараться максимально облегчить их работу и ускорить "вживание" в систему AutoCAD.

"Тетки"

Это особая разновидность "чайников", причем любого пола, хотя чаще всего это милые дамы бальзаковского возраста. От обычных "чайников" их отличает то, что они прекрасно знают предмет (проектирование), но в традиционных (бумажных) технологиях. Иногда это знание начинает вредить при переходе на работу в САПР. Например, "тетки" с трудом понимают концепцию рисования в натуральную величину (они всю жизнь начинали чертеж с рисования формата, а потом вписывали в него изображения реальных объектов). "Теток" можно сравнить с людьми, которых в зрелом возрасте впервые заставили кататься на коньках. "Тетки" обладают большой работоспособностью и выносливостью, но из-за возраста им просто трудно осваивать компьютер. "Теток" также надо любить, уважать, набираться у них опыта и стараться делать программы так, чтобы они были удобны и понятны даже для них.

"Обезьяна с гранатой"

Пишем в единственном числе, потому что в каждом коллективе почему-то всегда имеется один экземпляр (существо) такого класса. Можно сравнить с ребенком, который еще не умеет ходить, но уже умеет бегать, причем ноги его несут куда угодно. Основной признак — нельзя предугадать, что оно сделает с "гранатой". В роли гранаты чаще всего выступает мышь, которая в руках такой особи совершает произвольные передвижения со случайными нажатиями кнопок. После некоторой практики к мыши добавляются и случайные нажатия клавиш на клавиатуре. Обычно эту стадию развития проходят быстро (но смена всегда подрастает). Для разработчика "обезьяна с гранатой" является очень ценной добычей. Именно она будет проверять программы на прочность, т. е. на устойчивость к неправильным действиям пользователей.

"Нормальные пользователи"

Это бывшие "чайники", научившиеся работать. Нормального пользователя можно сравнить с повзрослевшим ребенком, который умеет и ходить, и бегать. При необ-

ходимости он может разобраться с многочисленными настройками AutoCAD и освоить любые новинки, но "нормальному пользователю" надо работать, а не изучать "секреты" и "внутренние миры" различных программ, которые для них не более чем инструмент. Таких пользователей большинство, и наши программы должны дать им возможность быстро и эффективно работать с минимальным отвлечением на ненужные исследования.

"Профи"

В отличие от "нормальных пользователей" они ненормальные. Количество таких экземпляров на любой территории ограничено. Главной целью "профи" считает всестороннее изучение любых попадающихся ему программ. Он постоянно что-то "закачивает", "ставит", "сносит". Систему AutoCAD он знает в совершенстве, но само рисование ему уже не интересно. Покопаться в "кишочках" у системы — другое дело. Обычно "профи" плохо кончают пользовательскую жизнь. Довольно быстро они трансформируются в программистов, заболевают посещением форумов в Интернете, а то и вообще превращаются в технических писателей и начинают писать книги, подобные той, которую вы читаете. Мы, при разработке *своей системы*, будем использовать колоссальную энергию "профи" в своих корыстных целях — именно они будут заниматься тестированием, профессиональной проверкой (в отличие от стихийной "обезьяны с гранатой") программ, изучением настроек и прочих секретов. Из таких специалистов мы и будем готовить администраторов *нашей системы*. В качестве приманки мы будем использовать специальные привилегии администраторов и некоторые программы класса "только для профи!".

"Ламеры"

Особый неприятный подвид амбициозных "чайников", возомнивших себя "профи". "Ламера" можно сравнить с ребенком, который уже научился ходить, но думает, что уже может бегать. Все родители знают, какой это опасный период, но если все дети взрослеют, то многие "ламеры" остаются такими на всю жизнь. Именно про "ламеров" ходят всякие программистские байки. Причем любят их рассказывать сами же "ламеры", с непременным щеголяньем компьютерным сленгом. Иногда "ламерами", совершенно неправильно, называют обычных добропорядочных "чайников". В *своей системе* эту категорию мы будем учитывать для того, чтобы обезопасить систему от безграмотных действий.

"Крутые"

Эта немногочисленная группа образуется в результате деградации отдельных "профи". Активная деятельность им надоедает, они впадают в апатию, начинают уже только ради денег быстро "клепать" проекты, нимало не заботясь об интересах окружающих. Внешним признаком является отсутствие в системе AutoCAD каких-либо панелей, меню, работа на 2—3 слоях, использование всего нескольких команд с вызовом через псевдонимы, отказ от каких-либо "примочек". Это состояние уже не поддается исправлению. Нам такие пользователи неинтересны, так же как неинтересны им наши программы.

Остальные группы ("сисадмины", "эникейщики", "мальчики по вызову", "негры", "начальники" и т. п.) значительного влияния на *нашу систему* не окажут, и мы их не будем рассматривать.

Как должна работать наша система

Наша система должна работать просто, удобно и надежно.

Просто, это как?

Просто, в нашем понимании, означает, что работа пользователя должна быть максимально приближена к работе в "просто AutoCAD", интерфейс наших команд должен иметь то же, что и стандартные команды:

- приглашения и опции командной строки;
- ввод значений с клавиатуры, из экранного и контекстного меню;
- значения по умолчанию.

В последних версиях системы AutoCAD интерфейс пользователя слишком усложнился. Конечно, появилась масса новых возможностей, но использовать их иногда очень непросто. Пользователей категории "профи" это не смущает, им только того и надо — позаниматься исследованием внутренностей AutoCAD. А вот "теткам" и "чайникам" приходится туго. Им, например, бывает сложно догадаться, что для получения требуемого эффекта надо на третьей вкладке второго диалогового окна настройки размерных стилей изменить состояние одного из десяти "переключателей".

В *нашей системе* все настройки должны выполняться скрытно от пользователя в зависимости от пары основных свойств рисунка, задаваемых при его создании. Далее пользователь должен работать про простой схеме — щелчок в меню для выбора команды, выбор параметров из максимально упрощенного диалога, несколько указаний на экране, завершение команды путем выбора явно видимой опции выхода.

Удобно, это как?

Удобство работы складывается из сотен незаметных, на первый взгляд, мелочей. Например, удобно для установки ширины линии выбрать из двух вариантов: "тонкая" или "основная" или, при необходимости, задать ширину в миллиметрах на бумаге, а не высчитывать ее в единицах рисунка с учетом масштабного коэффициента.

Удобно для отключения слоя указать на объект, а не выбирать слой (еще неизвестно какой именно) из длинного списка.

Удобно, щелкнув по кнопке "Такой же" и указав на любое изображение, сразу начать рисовать такое же изображение, на таком же слое. При этом программа сама определит, что если это блок, то надо запрашивать только новые точки вставки, но не спрашивать имя блока, масштаб и угол поворота, а если это полилиния, то не надо спрашивать ширину, но, возможно, ее необходимо замкнуть.

Удобно, когда в нужный (и только в нужный) момент программа сама включает необходимую объектную привязку или поворачивает на требуемый угол курсор.

Удобно, это когда пользователь, раздумав продолжать команду, имеет возможность в любой момент завершить ее штатными средствами, а не проходить все этапы ввода данных.

Все подобные "удобства" мы и должны реализовать при разработке системы.

Надежно, это как?

Надежность системы, прежде всего, означает защиту от ошибочных действий пользователя. Программы должны быть устойчивы как к беспорядочным щелчкам "обезьяны с гранатой", так и к непреднамеренным ошибкам добропорядочных "нормальных пользователей". Например, программа не должна допускать ввод нулевых или отрицательных значений, если это недопустимо по логике работы, должна еще при выборе объектов отбрасывать примитивы недопустимых типов, должна понимать, отказался пользователь от продолжения при выборе объекта, или просто "промазал".

Если пользователь прервал работу нажатием клавиши <Esc>, то программа должна поинтересоваться, не желает ли "клиент" отменить все сделанное, или он просто ошибся. Надежная программа, в любом случае, в том числе при ошибке, должна восстановить всю рабочую обстановку, действовавшую до ее запуска.

Конечно, добиться того, чтобы система одновременно работала просто, удобно и надежно, совсем не так-то легко. Как это сделать практически, мы будем рассматривать во всей книге.

Чем наша система должна отличаться от других

Любой программный продукт может иметь успех только в том случае, если он чем-то существенно отличается от аналогов. Хотя бы ценой. Вопрос о цене оставим пока открытым, рассмотрим технические аспекты.

Мы задумали *свою систему* для автоматизации работ многих разделов проекта. Это уже существенное отличие, а если мы сумеем реализовать задуманное, то это будет и существенным достоинством.

Множество решаемых задач предполагает наличие и множества команд (в терминологии AutoCAD), и множества пунктов меню или кнопок на панелях инструментов. Спрятать все это в падающих (иначе называемых ниспадающими, раскрывающимися или выпадающими) меню системы AutoCAD можно, но добираться до тысяч вложенных пунктов традиционного меню будет очень неудобно. Придется нам разработать собственный нестандартный интерфейс меню (сохраняя стандартный интерфейс команд). Это будет второе существенное отличие *нашей системы* от аналогов. Реализации задуманного интерфейса будет посвящена значительная часть книги.

Сколько бы мы не предусматривали команд, для такого количества охватываемых разделов их всегда будет мало. Самым лучшим выходом будет предоставление пользователям категории "профи" или прикладным программистам возможности расширения *нашей системы.* Нам все равно придется разработать множество библиотечных функций, так почему бы не предоставить возможность их использовать? Для этого их надо только документировать. Опыт прежних разработок показывает, что даже квалифицированная "тетка" вполне способна расширить личный сборник типовых текстов, "нормальный пользователь" вполне может включить в иллюстрированное меню коллекцию своих блоков, а уж "профи" без труда разработает автоматизацию рисования целого раздела, например, какого-нибудь специального технологического оборудования.

Возможность использования интерфейса API (Application Programming Interface) и будет третьим существенным отличием *нашей системы*.

Примечание

Критики нашего подхода могут заметить, что такая универсальная система будет напоминать утку, которая может и бегать, и летать, и плавать, но все это делает плохо. В специализированных САПР, мол, все решается лучше.

Нам больше нравится сравнение со взводом спецназа, в котором каждый боец и бегает, и стреляет, и плавает хуже, чем профессиональный спортсмен. Но комплекс любых боевых задач сплоченное подразделение решит лучше, чем сборная команда разных чемпионов мира. Так же, как трудно собрать такую гипотетическую сборную, так нелегко собрать и "сборную" специализированных САПР, совместимых между собой.

Какие задачи должна решать наша система

В результате предварительного анализа проектов и приемов работы мы уже имеем общее представление о том, что должна делать *наша система*. На следующем этапе необходимо систематизировать полученные сведения. На этом же этапе следует обзавестись всеми стандартами СПДС и ЕСКД, для того, чтобы точно, а не по памяти знать, как именно должны выглядеть изображения, которые мы будем создавать. Очень часто опытные проектировщики "вдруг" узнают, что какие-то элементы давно уже рисуются не так, как они привыкли. Наши программы, если мы их сделаем неверно, будут многократно тиражировать наши ошибки. Встречаются примеры, когда в распространенных и сертифицированных программах имеются ошибки именно из-за элементарного несоответствия ГОСТ.

Итак, приступаем к систематизации задач. Напоминаем, что все надо документировать.

Замечание

В концепции Extreme Programming, о которой мы упоминали в *главе 1*, описания того, как система должна работать, называются Users Story. Считается, что Users Story готовятся заказчиком, записываются на специальных карточках и являются основой и для планирования, и для программирования. Теоретически это безупречно, но попробуйте затребовать у нашего заказчика эти самые "истории"! В лучшем случае вам попытаются что-то объяснить "на пальцах", обычный ответ будет: "Вам надо, вы и делайте". Вот мы и будем "делать"[1].

Составление перечня общих задач

Многие задачи являются общими для всех разделов проекта. Типичные примеры:

- рисование форматов, различных ведомостей для общих данных по рабочим чертежам, написание текстов, позиций, обозначение разрезов, узлов и т. п.;

[1] Если "где-то на Западе" программирование с использованием Users Story считается "экстремальным", то с точки зрения наших разработчиков это работа в тепличных условиях, а "русский экстрим" авторам Extreme Programming просто не понять.

- рисование линий, контуров и других объектов, встречающихся во всех разделах — прямоугольники, кольца, "пятна", штриховки и заливки и т. п.;
- специфичные для AutoCAD вспомогательные программы, облегчающие, по сравнению со стандартными средствами, работу пользователя — управление слоями, изображением, типами линий, штриховками и т. п.

Многие общие задачи решаются с помощью сборников утилит, таких как Express Tools или ToolPac. Это замечательные пакеты, но рассчитывать только на их использование нельзя. В *своей системе* мы можем сделать почти все такие же "вкусности", но более простыми методами, доступными и пользователям категории "тетки".

Все задачи общего характера надо заносить в специальный реестр, сразу же намечая возможные способы решения. Это и будут "Russian Users Story". Реестр лучше всего сделать в виде базы данных MS Access. В такой базе можно хранить несколько таблиц различного назначения. Если вы не поленитесь сделать это (надо затратить минут 15 для создания структуры таблицы и формы представления данных), то получите своеобразную базу знаний, которая вам неоднократно пригодится (рис. 2.1 и 2.2). В случае необходимости вы сможете экспортировать данные в любой формат.

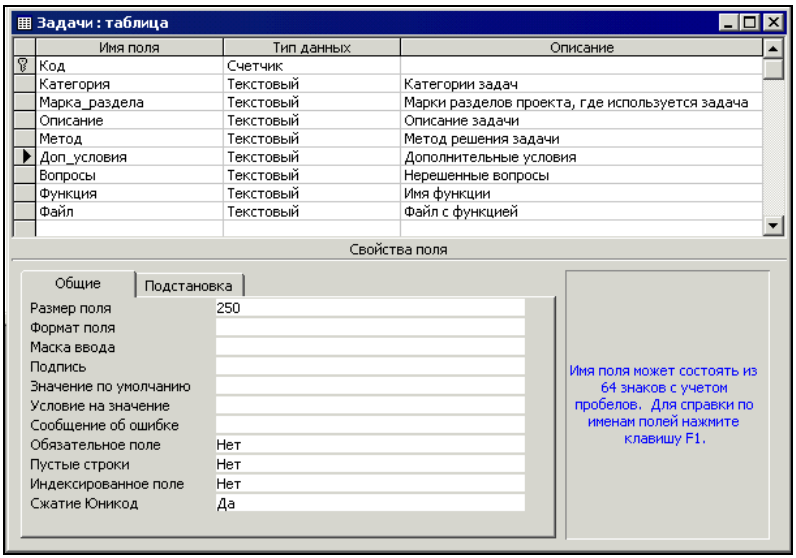


Рис. 2.1. Структура таблицы с перечнем задач

Вводя информацию о задачах в таблицу, вы сразу же начнете задумываться о том, как объединить задачи, хотя бы потому, что вам будет просто лень вводить много однотипной информации. Как только вы в третий раз занесете в таблицу очередную "Ведомость...", родится идея, что рисование всех ведомостей следует объединить в одну задачу.

После ввода списка задач их можно отсортировать по различным полям, найти подобные, объединить, в процессе разработки заносить дополнительные данные, на-

пример имя функции и имя файла, в котором эта функция определена. Все это очень пригодится в дальнейшей работе.

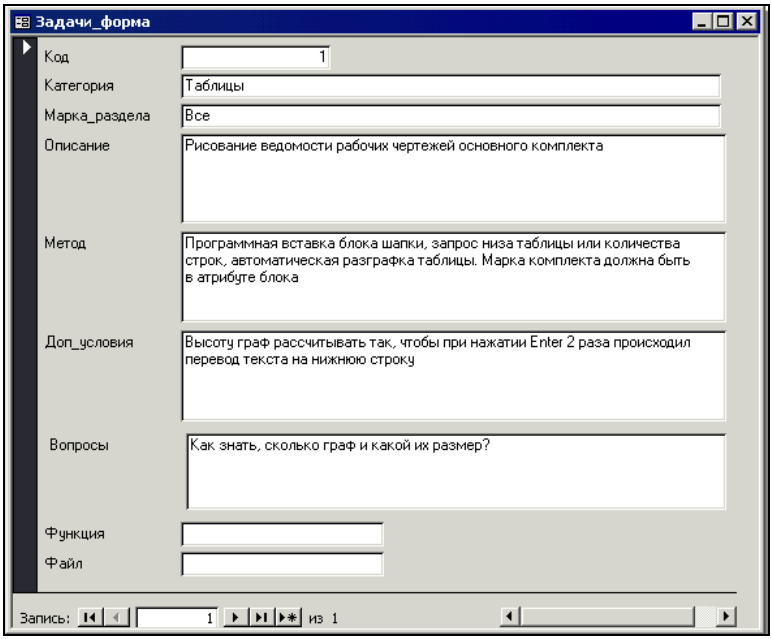


Рис. 2.2. Форма ввода данных таблицы задач

Конечно, вы можете игнорировать наши советы и держать реестр задач "в уме". Это тоже хороший способ сойти с ума, рано или поздно.

Напоминание

Не забывайте фиксировать планируемые задачи в рабочем плане, о котором мы говорили в *главе 1*.

Составление перечня специальных задач

Специальных задач наберется очень много. Только различных видов изображений можно насчитать до 10 тысяч. Специальные задачи следует занести в базу данных и также пытаться их объединять. При составлении перечня специальных задач необходимо установить приоритет их разработки. При решении этого вопроса нужно очень взвешенно рассматривать просьбы отдельных пользователей. В определенный момент времени конкретному пользователю может понадобиться очень сложная задача, на разработку которой потребуется значительное время. Поддавшись мольбе пользователя, вы можете просто увязнуть в длительной разработке, отложив другие программы. А потом выяснится, что эта задача вообще никому другому не нужна или нужна раз в 10 лет. Необходимо в первую очередь разработать программы мас-

сового применения, выделив из них те, реализация которых не займет слишком много времени. Более сложные задачи следует запланировать на *последующие версии* системы.

Как превратить специальные задачи в общие

Специальные задачи нужно стремиться превратить в общие. Поясним это на примере. В каждом разделе проекта требуется рисование различных таблиц. В строительной части это может быть "Ведомость проемов ворот и дверей" или "Таблица отправочных марок", в чертежах марки ОВ (отопление, вентиляция и кондиционирование воздуха) — "Характеристика отопительно-вентиляционных систем" (да еще в нескольких вариантах: с воздухонагревателями, с воздухоохладителями, с фильтрами или без них), в чертежах ТХ (технология производства) — "Ведомость технологических трубопроводов" и т. д. Для некоторых разделов требуется рисование таблиц с вертикальным расположением шапки, например заготовок продольных профилей. В некоторых шапках может быть атрибут с переменным значением.

Для того чтобы не разрабатывать много однотипных программ, следует объединить все эти задачи в одну. Можно заранее наметить разработку гипотетической функции, имеющей примерно такой формат вызова:

```
(РИСОВАТЬ_ТАБЛИЦУ <БЛОК_ШАПКИ> <АТРИБУТ_ШАПКИ|NIL> <ВЕРТИКАЛЬНАЯ?>)
```

Для таблиц с изменяемой концовкой шапки можно, конечно, сделать индивидуальные программы (запрос варианта, позиционирование заголовка и т. п.), но проще сделать несколько блоков шапок и свести частные задачи к общей.

Такой подход позволяет значительно сократить объем программирования, упростить отладку и сопровождение.

Примеры таких общих задач приведены в *части III*.

Как определить методы решения задач

На этапе постановки задачи необходимо определиться с методикой решения. Для некоторых задач могут использоваться довольно хитрые специальные алгоритмы. Это относится не только к выполнению специальных расчетов, но и к простому рисованию.

Типичный простейший пример — рисование прямоугольных предметов. Для рисования прямоугольников командой RECTANG под различными углами требуется поворот системы координат, рисование всегда начинается с угла, а иногда необходимо задать центр прямоугольника. Кроме "чистых" прямоугольников часто рисуют оборудование прямоугольной формы. В этом случае можно создать блоки для каждого типоразмера стола или шкафа, а можно иметь единичные блоки и масштабировать их при вставке (разумеется, программным путем). Не столь очевидным является то, что вставлять с разными масштабами по осям X и Y можно не только прямоугольники, но и любые изображения, вписываемые в прямоугольники (например, крутоизогнутый отвод), причем один блок может использоваться в разных разделах, для рисования совершенно различных предметов. Конечно, здесь нужен разумный под-

ход, т. к. изображение предмета со значительной разницей масштабов по осям может оказаться слишком искаженным.

Еще один пример — рисование радиальных (центробежных) вентиляторов. Имеется много моделей и типоразмеров вентиляторов, каждый из которых можно сделать в виде блока. Таких блоков потребуется сотни, но если знать, что геометрические размеры "улитки" вентилятора являются функциями диаметра рабочего колеса, то множество блоков можно заменить одной программой.

Во время разработки методов решения задач важно учитывать удобство работы пользователя. Часто это относится к порядку ввода данных. Об этом мы уже говорили, когда советовали ознакомиться со спецификой работы пользователей.

Вот еще один простой пример. Патрубок (трубопровод с фланцем) можно начать рисовать или от зеркала фланца, или от конца патрубка. Иногда требуется изобразить фланцы на обоих концах трубопровода ("катушка"). Кроме того, эти детали обычно рисуются не сами по себе, а в составе системы трубопроводов, т. е. может потребоваться изобразить, например, цепочку деталей "трубопровод — переход — трубопровод — фланец — задвижка — фланец — подъем — отвод — трубопровод — фланец". Было бы удобно при рисовании каждого последующего элемента автоматически привязываться к концу предыдущего элемента, но в то же время иметь возможность начать рисование любого элемента трубопровода со вновь задаваемой точки. Если чуть-чуть подумать, то окажется, что реализовать рисование всех узлов трубопроводов можно с помощью одной гипотетической (пока) функции:

```
(ТРУБА <Имя_элемента> <функция> <Подсказка_1> <Подсказка_2> <Два_диаметра?>
<Меню_опций>)
```

Если еще подумать, то окажется, что с помощью этой функции можно рисовать и элементы вентиляционных систем, а если очень крепко призадуматься, то она окажется пригодной и для рисования различных наружных и внутренних коммуникаций.

Замечание

У квалифицированных читателей может возникнуть вопрос о том, почему авторы столь неопределенно говорят о методике постановки задач, придумывают какие-то "реестры задач" и другие дилетантские методы, когда существуют, пусть устаревшие, стандарты, в которых четко определены и термины, и требования к содержанию и оформлению всех видов программных документов.

Ответ прост — прочитайте еще раз введение к нашей книге и начало *главы 1*.

Книга предназначена для специалистов, которые *вынуждены* заниматься разработками в условиях ограничения (или полного отсутствия) средств и времени. У них просто нет возможности буквально следовать требованиям стандартов и оформлять надлежащим образом программные документы. Оформить-то можно, но конечного результата может уже не получиться. Мы неоднократно видели многотомные проекты автоматизированных систем, на разработку которых (проектов, а не самих систем) были затрачены огромные средства. А систем, которые должны были быть реализованы по этим проектам, не видели.

В этой книге мы пытаемся хоть как-то помочь разработчикам (пусть иногда "шаманскими" методами), которые часто вообще считают, что разработка программ должна начинаться с набивки исходного текста в редакторе.

Как придумать "имя собственное" для нашей системы

Внимательный читатель давно заметил, что до сих пор мы использовали весьма неопределенное название "наша система", выделяя его курсивом. Теперь настало время придумать имя для *нашей системы*.

Краткое "имя собственное" программного продукта имеет огромное значение. Название продукта должно быть запоминающимся, по возможности кратким, оригинальным, легко произносимым, по возможности отражая назначение продукта. Хорошее название, например, позволяет *начальнику* легко запомнить его и, при случае, где-нибудь показать свою образованность. AutoCAD — отличное название, а Plant-4D, на наш взгляд, неудачное, потому что его (без прохождения специального курса обучения) не сможет правильно произнести и запомнить *ваш директор*. Иногда вообще названия произносятся не так, как пишутся. Типичный пример — интерфейс SCSI, название которого произносится как "скази".

Замечание

Несколько лет назад мы изменили название одного из своих продуктов с труднопроизносимого zCAD на InCAD и сразу почувствовали, что он стал лучше восприниматься "распорядителями кредитов". Версия для Windows, получившая название "BestIA" (именно с таким, не расшифровываемым написанием), стала расходиться еще лучше, особенно в силовых структурах, т. к. людям в погонах слово "бестия" почему-то очень приглянулось. *(Сергей Зуев)*

Превращение "нашей системы" в "ruCAD"

В книге мы выполняем условную разработку условного продукта. Мы тут посоветовались и сообщаем вам, что есть мнение присвоить разрабатываемой системе название

.ru/CAD

Когда система станет популярной, то наемные имиджмейкеры придумают красивую легенду о происхождении этого (или другого) названия, а пока мы можем сказать правду:

- **CAD** — логичное название системы, являющейся приложением к AutoCAD;
- **.ru** (точка ру) — намек на то, что корни нашей системы произрастают из российского сегмента Интернета;
- **/** — (обратный слэш), с одной стороны, позволяет любому каталогу с именем CAD в домене ru как бы рекламировать нашу систему; с другой стороны, подобные сочетания (слэши, разные регистры символов) придают имени некий "кулхацкерский" оттенок, что может привлечь дополнительное внимание молодежи.

Пусть **.ru/CAD** будет "большим", официальным именем, для повседневного же употребления мы будем называть нашу систему проще — **ruCAD**, а фамильярное маленькое **ru** мы будем применять там, где больше ничего и не вписать, например в иконках.

Логотип

Теперь нам непременно нужен логотип, без которого невозможно представить программный продукт. Разработка хорошего логотипа очень сложное дело. Когда-нибудь мы закажем его лучшим профессионалам, а пока поступим следующим образом:

- "сфотографируем" на сканере руку подходящей "тетки";
- обработаем изображение в Adobe Photoshop (жуткие подробности пропускаем), скомбинируем его с текстом и получим логотип (рис. 2.3).

Рис. 2.3. Логотип **.ru/CAD**

Логотип получился в меру пошлым (избитое изображение руки, текст с тенью), но "технологичным", т. к. очень легко может послужить основой для заставок, баннеров и других изображений с дополнительными текстами на поле, которое мы для этого и предусмотрели под надписью.

Конечно, все это не надо воспринимать слишком серьезно, и название и логотип всего лишь рабочие. Вы в своих системах подойдете к этим вопросам с большей ответственностью.

Для оформления технического задания и других документов мы "зарегистрируем" мифическое предприятие с названием "НПО РУКАД ГРОУП". Такое название директорам старого образца будет намекать на некое "научно-производственное объединение" (на самом же деле это просто набор символов), а "иностранное" слово ГРОУП (group) будет вызывать ассоциации с солидными фирмами, не стесняющимися регистрировать "русские" названия (Майкрософт, Аутодеск и т. п.).

Заодно мы придумаем и мифического заказчика — "Проектно-строительное объединение АрбатовСтройПроект". Просто для того, чтобы был какой-то пример.

Стадийность разработки

В соответствии с ГОСТ 19.102-77 предусматривались следующие стадии разработки программ и программной документации.

Техническое задание

В стадию технического задания (ТЗ) включают следующие этапы:

- обоснование необходимости разработки программы;
- научно-исследовательские работы:
  - разработка и утверждение технического задания;
  - постановка задачи, сбор исходных материалов;

- определение структуры входных и выходных данных;
- предварительный выбор методов решения задач;
- обоснование целесообразности применения ранее разработанных программ;
- определение требований к техническим средствам;
- обоснование принципиальной возможности решения поставленной задачи;
- определение требований к программе;
- технико-экономическое обоснование разработки программы;
- определение стадий, этапов и сроков разработки программы и документации на нее;
- выбор языков программирования;
- определение необходимости проведения научно-исследовательских работ на последующих стадиях;

❒ согласование и утверждение технического задания.

Эскизный проект

В стадию эскизного проекта включаются следующие работы:

❒ предварительная разработка структуры входных и выходных данных;

❒ уточнение методов решения задачи;

❒ разработка общего описания алгоритма решения задач;

❒ разработка технико-экономического обоснования;

❒ разработка пояснительной записки;

❒ согласование и утверждение эскизного проекта.

Технический проект

На стадии технического проекта выполняют такие работы:

❒ уточнение структуры входных и выходных данных;

❒ разработка алгоритма решения задачи;

❒ определение формы представления входных и выходных данных;

❒ разработка структуры программы;

❒ окончательное определение конфигурации технических средств;

❒ составление плана мероприятий по разработке и внедрению программы;

❒ разработка пояснительной записки;

❒ согласование и утверждение технического проекта.

Рабочий проект

На стадии рабочего проекта выполняются следующие работы:

❒ разработка программы;

❒ разработка программной документации;

- ❒ испытания программы;
- ❒ программирование и отладка программы;
- ❒ изготовление программы-оригинала;
- ❒ разработка программных документов;
- ❒ разработка, согласование и утверждение порядка и методики испытаний;
- ❒ проведение испытаний;
- ❒ корректировка программы и программной документации по результатам испытаний.

Внедрение

На стадии внедрения выполняются следующие работы:

- ❒ подготовка и передача программы и программной документации;
- ❒ оформление и утверждение акта о передаче программы;
- ❒ передача программы в фонд алгоритмов и программ.

Мы уже неоднократно говорили о том, что в современных условиях такая многостадийная разработка просто невозможна. Конечно, этим увлекательным делом можно было бы заняться, если бы нас интересовал не конечный результат, а "освоение средств". Умные люди понимали это и в те далекие годы, поэтому они включили в ГОСТ скромное примечание о том, что допускается исключать стадии эскизного и технического проекта (с указанием этого в техническом задании). Воспользуемся данной лазейкой, чтобы наше техническое задание даже формально соответствовало ГОСТ. А про требование о "передаче программы в фонд алгоритмов и программ" вообще забудем. Держите карман шире!

Подготовка окончательного текста технического задания

Теперь, после проведения работ по анализу задач, мы готовы составить окончательный текст технического задания. При написании задания мы будем придерживаться структуры, рекомендуемой ГОСТ 19.201-78. Курсивом будем выделять наши комментарии, не входящие в текст задания.

Текст технического задания должен включать в себя следующие разделы.

Титульная часть

- ❒ Лист утверждения (не входит в общее количество листов документа).
- ❒ Титульный лист (первый лист документа).

Информационная часть

- ❒ Аннотация.
- ❒ Лист содержания.

Материалы титульной части, выполняемые по ГОСТ 19.106-78, для сокращения места мы в книге не приводим, но разработчик должен помнить, что именно там располагается подпись Самого Главного Начальника, благословляющего дальнейшую деятельность (и заодно принимающего на себя формальную ответственность за все, что мы собрались натворить). Пример:

Техническое задание
на разработку программного изделия

Наименование

Первая очередь интегрированной системы автоматизированного проектирования ruCAD.

Обратите внимание: пишем "Первая очередь", что дает возможность говорить о том, что какие-то виды работ будут автоматизироваться НЕ в первую очередь. Кроме того, мы официально называем систему "интегрированной", что может очень пригодиться в будущем.

Область применения

Строительное проектирование промышленных и гражданских зданий и сооружений.

Мы устанавливаем очень широкую область применения. В определенной степени это хорошо, т. к. уже на стадии ТЗ мы пытаемся узаконить захват большой сферы применения.

Основание для разработки

Программа информатизации проектно-строительного объединения АрбатовСтройПроект на 2003—2010 годы, утвержденная Администрацией г. Арбатов (постановление от 01 апреля 2002 г. № 238).

Если есть подобный документ, его надо указать. Еще лучше, если их несколько. Это забота заказчика, т. к. он (в случае чего) должен обосновывать, почему он тратит деньги на ЭТО. В прежние времена никто не мог что-то сделать на основании здравого смысла, обязательно требовалось "Основание" в виде какого-то документа. Для этого пункта может сойти и какая-нибудь "Программа развития ... до 2005 года", наверняка принимавшаяся в годы перестройки.

Назначение разработки

- ❐ Автоматизация выполнения рабочих чертежей основных комплектов проектной документации:

  Мы пишем "выполнения рабочих чертежей", а не "проектирования", чтобы к нашей системе не предъявлялись такие же требования, как к САПР высокого уровня. За такой фразой и скрывается "чертилка" или "электронный кульман".

  - Генеральный план и транспорт.
  - Архитектурно-строительная часть, включая архитектурные решения и рабочие чертежи конструкций (железобетонные, каменные, металлические, деревянные).

- Технология производства по номенклатуре работ АрбатовСтройПроект (торгово-технологическое оборудование, жилые и общественные здания, котельные, тепловые пункты).
- Отопление, вентиляция и кондиционирование воздуха.
- Водопровод и канализация.
- Электрическое освещение.
- Силовое электрооборудование.
- Слаботочные устройства.
- Автоматизация технологических процессов и санитарно-технических систем.
- Наружные инженерные сети (теплоснабжение сети, водоснабжение, канализация, наружные газопроводы, технологические трубопроводы).

Мы честно берем на себя обязательство по автоматизации черчения практически всех разделов, но ограничиваем технологию производства, которая может быть самой разнообразной.

- ❐ Предоставление интерфейсов прикладного программиста для адаптации и расширения системы ruCAD.

  Это очень важный пункт, обязывающий разработчиков, как минимум, написать хорошее Руководство программиста. Фактически мы дадим возможность дополнить ruCAD всем, что не успеем сделать сами.

- ❐ Внедрение элементов документооборота и стандартов автоматизированного проектирования в ПСО АрбатовСтройПроект.

  Тоже важный пункт. Стандарты проектирования и, как минимум, стандарты хранения данных, скромно называемые элементами документооборота, очень нужны. Включение этих понятий в Утверждаемый документ дает разработчикам дополнительные полномочия.

Заказчик

Проектно-строительное объединение АрбатовСтройПроект.

Этого пункта нет в ГОСТ, мы его вводим сами. Пусть будет четко написано и утверждено, кто же оплачивает и принимает работы.

Исполнитель

НПО РУКАД ГРОУП, Арбатовский филиал Черноморского отделения.

Этого пункта также нет в ГОСТ, мы его вводим сами. Пусть будет лишний раз указан исполнитель, потому что непременно найдутся "настоящие системные интеграторы", которые будут просить отдать работу им.

Технические требования к программному изделию

Очень важный раздел, устанавливающий основные параметры разрабатываемого программного изделия, которые мы уже обсудили. Теперь их надо сформулировать, по возможности кратко и обобщенно.

Требования к функциональным характеристикам

В подразделе "Требования к функциональным характеристикам" должны быть указаны требования к составу выполняемых функций, организации входных и выходных данных, временным характеристикам и т. п.

Программное изделие должно обеспечивать следующие возможности.

- Вычерчивание общих данных, планов, разрезов, схем, узлов, чертежей установок и других изображений, предусмотренных стандартами системы проектной документации для строительства (СПДС) для разделов (марок) рабочих чертежей, указанных в разделе "Назначение разработки".
- Организацию коллективной работы над проектами, с обменом данными между исполнителями смежных разделов в электронном виде.
- Совместимость документов, разрабатываемых в составе смежных разделов.
- Дополнение специалистами заказчика библиотек изображений оборудования и изделий, а также баз данных, используемых при составлении спецификаций оборудования.
- Дополнение специалистами заказчика библиотек программ для решения специализированных задач с использованием интерфейсов программного изделия.

Требования к надежности

Программное изделие должно надежно функционировать в операционной системе и базовом графическом редакторе AutoCAD, не снижая их собственных показателей надежности.

Программное изделие должно быть устойчиво к ошибочным действиям пользователя, осуществлять контроль входной и выходной информации, восстанавливать параметры окружения при возникновении сбойных ситуаций.

Условия эксплуатации

Программное изделие должно функционировать в условиях эксплуатации, соответствующих нормативным требованиям, предъявляемых к зданиям управлений.

Программное обеспечение должно обеспечивать возможность эксплуатации пользователями, имеющими в привилегии не выше чем для базовой программной системы AutoCAD, за исключением отдельных действий (установка, настройка), для которых требуются привилегии Администратора.

К сожалению, постоянные изменения структуры прав групп пользователей в различных версиях операционных систем Windows NT и не самый рациональный подход фирмы Autodesk к работе с реестром, нередко приводит к тому, что системный администратор вводит специальные правила для пользователей системы AutoCAD. А потому, чем меньше привилегий потребуется для нормальной работы системы от имени пользователя, тем лучше.

Требования к составу и параметрам технических средств

Параметры технических средств для эксплуатации программного изделия не должны превышать требуемых параметров для базовой системы AutoCAD.

Требования к информационной и программной совместимости

- Программное изделие должно быть совместимым с операционными системами Windows 98, Windows ME, Windows 2000 и Windows XP.
- Программное изделие должно быть совместимым с системой AutoCAD 2002. Во второй очереди разработки должна обеспечиваться совместимость с системой AutoCAD 2004[1].

  В марте 2003 года поступила в продажу система AutoCAD 2004. Эта версия наверняка будет популярной, но пользователям потребуется определенное время на переход. Наши программы будут учитывать возможности и особенности этой новой версии, но мы также зарезервируем некоторое время на возможные непредвиденные ситуации.
- Программное изделие должно обеспечивать возможность обмена данными с другими программными изделиями через текстовые файлы документированных или стандартных форматов (INI, XML, CSV).

Требования к маркировке и упаковке

Программное изделие должно поставляться на магнитных носителях CD-ROM. Программная документация должна поставляться в электронном виде в форматах, обеспечивающих ее чтение средствами операционной системы или свободно распространяемыми программными продуктами.

Допускается поставка магнитных носителей в стандартной упаковке для CD-ROM с поясняющими вкладышами и нанесением на нерабочую сторону диска только идентификационных меток.

Требования к транспортированию и хранению

Транспортировка и хранение магнитных носителей с программным изделием...

В подразделе "Требования к транспортированию и хранению" для программного изделия должны быть указаны условия транспортирования, места хранения, условия хранения, условия складирования, сроки хранения в различных условиях.

Специальные требования

В этот подраздел заказчик и исполнитель могут вписывать другие требуемые условия. К таким или подобным (например, "Особые условия") разделам надо относиться очень осторожно, т. к. здесь могут быть спрятаны слишком "хитрые" требования.

Внедрение заказчиком других систем автоматизированного проектирования и систем электронного документооборота по иным договорам производится с учетом рекомендаций разработчиков настоящего программного изделия.

Это очень важное требование. Какому-нибудь отделу может понравиться совсем другая САПР, в том числе совершенно несовместимая. Даже если такая САПР базируется на AutoCAD, она может функционировать на других принципах, например, выполнять

[1] В марте 2004 года выпущена версия AutoCAD 2005, но в тексте задания мы о ней не упоминаем — для перехода на эту версию мы можем "выбить" новое задание.

рисунки не в натуральном масштабе. Утверждающая инстанция, если уж поручает нам разработать САПР для всех разделов проекта, должна предоставить возможность достаточно жестко влиять на техническую политику. Это совершенно не означает, что мы должны непременно отвергать всех конкурентов. Большинство САПР на базе системы AutoCAD совместимы между собой, но встречаются и досадные отклонения.

Технико-экономические показатели

В разделе "Технико-экономические показатели" должны быть указаны: ориентировочная экономическая эффективность, предполагаемая годовая потребность, экономические преимущества разработки по сравнению с лучшими отечественными и зарубежными образцами или аналогами. Это любимые пункты времен начала перестройки, особенно сравнение с "лучшими отечественными и зарубежными образцами или аналогами", показатели которых неизвестно откуда можно было взять. Постараемся написать проще.

Программное изделие должно обеспечить повышение производительности труда по разработке рабочих чертежей не менее чем в 4 раза, по сравнению с использованием только стандартных средств AutoCAD.

"В 4 раза" мы берем с потолка, можно обеспечить повышение производительности и в 50 раз, если бы выполнялись однотипные работы. Потери производительности происходят не столько на этапе черчения, сколько на предпроектных этапах, согласованиях и т. п. Мы вновь пишем о "разработке чертежей", а не о "проектировании", т. к. это разные понятия.

Стадии и этапы разработки

Программное изделие разрабатывается в одну стадию — рабочий проект. Сроки разработки устанавливаются Договором на разработку программного изделия.

Разработка в одну стадию позволяет значительно сократить объем практически ненужной документации. Мы исключаем стадии эскизного и технического проекта, но берем обязательство предоставить заказчику готовое программное изделие.

Порядок контроля и приемки

Контроль и приемка программного изделия осуществляются поэтапно:

1. Предварительные испытания отдельных компонентов программного изделия в процессе разработки, выполняющиеся на выделенных заказчиком рабочих местах.
2. Пробная эксплуатация всего программного изделия в реальных условиях по отдельному графику.
3. Окончательные испытания и приемка в промышленную эксплуатацию.

В процессе предварительных испытаний и пробной эксплуатации специалисты заказчика предоставляют исполнителю в электронном виде информацию о сбойных ситуациях и условиях их возникновения, а также предложения по совершенствованию программных средств.

Приемка программного изделия в промышленную эксплуатацию оформляется Актом приемки.

Приложения

В приложениях к техническому заданию, при необходимости (например, придать документу солидный, наукообразный вид), приводят схемы алгоритмов, таблицы, описания, обоснования, расчеты и другие документы, которые возможно использовать при разработке. Это могут быть как специальные методики, предоставляемые заказчиком, так и наши собственные материалы. Если бы вы прислушались к нашему совету создать базу данных перечня задач, то теперь могли бы быстро сформировать из нее отчет, который был бы прекрасным приложением к техническому заданию.

Подписи и согласования

Любой документ, тем более такой важный, как техническое задание, должен быть подписан разработчиками, согласован и утвержден. Хотя разработку технического задания выполняли мы с вами, подписан он должен быть представителем заказчика, имеющим достаточные полномочия, например главным инженером.

Разработчики программного изделия должны согласовать техническое задание. Желательно, чтобы согласования были проведены и с другими представителями заказчика, например с начальниками отделов, в которых будет использоваться система, и с руководителем "компьютерной" службы. Это позволит избежать последующих заявлений о том, что "все надо было делать не так". Подписи можно разместить в конце документа или на титульном листе, а при большом количестве согласований их можно вынести на специальный лист согласований.

ГЛАВА 3

Формирование базовых принципов

В этой главе мы постараемся, основываясь на утвержденном техническом задании, сформулировать основные принципы, которыми будем руководствоваться при разработке ruCAD. На этом этапе мы должны уже более конкретно подходить к решению вопросов, обозначенных на этапе подготовки технического задания.

Как запускать AutoCAD

Запуск AutoCAD чаще всего предусматривается с помощью ярлыков на рабочем столе или соответствующих пунктов в главном меню Windows. В свойствах ярлыка указываются рабочий каталог, полное имя файла acad.exe и один или несколько параметров командной строки.

AutoCAD имеет много параметров командной строки, приводимых в табл. 3.1. В числе прочих можно передавать и имя загружаемого рисунка.

Таблица 3.1. *Параметры командной строки AutoCAD*

Переключатель	Наименование	Примечание
/b	Имя пакетного файла	Указывает пакетный файл для его выполнения сразу после запуска AutoCAD. Пакетные файлы могут устанавливать параметры для нового рисунка. Файл должен иметь расширение scr
/t	Имя файла шаблона	Создает новый рисунок на основе шаблона. Файл должен иметь расширение dwt
/c	Папка конфигурации	Указывает путь к файлу конфигурации, который должен использоваться. Можно задавать здесь папку или конкретное имя файла. Файл должен иметь расширение cfg. Если переключатель /c не указан, то AutoCAD выполняет поиск в папке самого приложения AutoCAD и для определения файла конфигурации и пути к нему использует переменные среды ACADCFGW или ACADCFG

Таблица 3.1 *(окончание)*

Переключатель	Наименование	Примечание
/v	Имя вида	Определяет вид рисунка, отображаемый после запуска AutoCAD
/s	Папки вспомогательных файлов	Задает папки вспомогательных файлов, отличные от рабочей. Во вспомогательных файлах хранятся текстовые шрифты, меню, программы, типы линий и образцы штриховок. Допускается указывать до 15 папок. Имена папок должны разделяться точкой с запятой
/r	Системное устройство указания, принятое по умолчанию	Восстанавливает системное устройство указания, принятое по умолчанию. Создается новый файл конфигурации (acad2002.cfg), а предыдущая версия файла acad2002.cfg переименовывается в acad2002.bak
/nologo	Отключение логотипа AutoCAD на экране	Запускает AutoCAD без показа логотипа на экране
/p	Профиль, заданный пользователем, для запуска AutoCAD	Задает файл профиля, определенный пользователем, для запуска AutoCAD. Указанный профиль действует только в текущем сеансе AutoCAD. Установить профиль для всех сеансов можно в диалоговом окне **Options** (Настройка). Для этого предназначена вкладка **Profiles** (Профили), где можно создавать и импортировать файлы профилей. Профили, указанные опцией /p, отображаются в соответствующем списке диалогового окна **Options** (Настройка). Если указанный профиль не создан, то AutoCAD использует текущий профиль

Командная строка вызова программы с использованием переключателей выглядит следующим образом:

```
"устройство:путь\acad.exe" ["имя рисунка"] [/переключатель "имя"]
```

За каждым переключателем должен следовать символ пробела, затем имя файла, путь или имя вида в кавычках. Следующий пример показывает командную строку запуска AutoCAD из папки c:\Acad\2002, без заставки, с профилем из файла c:\.ru\CAD\ruCAD.arg с загрузкой файла c:\.ru\cad\samples\dwg\Сантехника\Водомеры.dwg:

```
"C:\Acad\2002\acad.exe" "c:\.ru\cad\samples\dwg\Сантехника\Водомеры.dwg" /nologo
/p "C:\.ru\CAD\ruCAD.arg"
```

AutoCAD устанавливает настройки среды в следующем порядке:

- если параметр (переключатель) задается в командной строке, то указанное значение имеет приоритет перед установленными в диалоговом окне **Options** (Настройка) и в соответствующей переменной среды;
- если параметр отсутствует в командной строке, используется значение соответствующего параметра диалогового окна **Options** (Настройка);
- если параметр не указан ни в командной строке, ни в диалоговом окне **Options** (Настройка), то используется значение соответствующей переменной среды.

Замечание

Переключатели командной строки и переменные среды переопределяют параметры диалогового окна **Options** (Настройка) только для текущего сеанса, не сохраняя изменений.

Интересен запуск с переключателем /c, указывающим на файл конфигурации, например:

```
"C:\Acad\2002\acad.exe" "c:\.ru\cad\samples\dwg\Сантехника\Водомеры.dwg" /nologo
/p "C:\.ru\CAD\ruCAD.arg" /c "C:\.ru\CAD\ruCAD.cfg"
```

В этом случае мы получаем возможность сохранять данные в файле конфигурации с помощью функций `setcfg` и `getcfg`, причем не в стандартном файле конфигурации AutoCAD, а в собственном файле в собственном каталоге. Таким образом, мы можем использовать файл конфигурации в качестве INI-файла без посторонних функций. Недостаток заключается в том, что файл конфигурации реально записывается на диск только после выхода из AutoCAD, а посторонние программы могут получать из него устаревшие данные.

При запуске системы AutoCAD с помощью ярлыка, в котором указан профиль, обычно происходит автоматическая загрузка требуемых библиотек и приложений, формируется необходимое меню и выполняются прочие требуемые настройки. При необходимости пользователь может создать собственные ярлыки, например для каждого рисунка.

В некоторых приложениях к AutoCAD используется иной способ — запуск AutoCAD из специальной программы (стартера), которая выполняет ряд действий до того, как AutoCAD будет запущен. В своей системе мы воспользуемся такой методикой, для чего разработаем специальное приложение-стартер.

На программу-стартер мы возложим следующие функции:

- идентификация пользователя, установка его предпочтений, привилегий и разрешений;
- выполнение основных настроек системы и реестра пользователей;
- выбор рабочей версии AutoCAD (возможно, на компьютере их имеется несколько);
- запуск AutoCAD с необходимыми параметрами командной строки (возможно с предварительным выбором документов для загрузки) и генерированием файла для автоматической загрузки требуемых библиотек.

Кроме того, стартер будет выступать в качестве внешнего COM-сервера, к объектам и методам которого можно обращаться из LISP-программ.

На начальном этапе, до разработки стартера, мы будем запускать систему AutoCAD с помощью ярлыка, отрабатывая параметры командной строки.

Как использовать шаблоны рисунков

Новый рисунок в AutoCAD можно создать с использованием шаблона. Шаблоном может быть рисунок, имеющий расширение dwt и хранящийся в каталоге шаблонов текущего профиля[1]. При использовании шаблона отпадает необходимость в ручной

[1] До версии AutoCAD R14 вместо *шаблонов* использовались *прототипы* рисунка, имевшие обычное расширение dwg.

настройке многочисленных параметров. Кроме того, в шаблоне могут быть уже нарисованы стандартные фрагменты (формат, основная надпись), предусмотрены требуемые видовые экраны и пр.

Шаблоны действительно являются очень удобным средством стандартизации чертежей в рамках рабочей группы. Но при достаточном разнообразии решаемых задач требуется и много шаблонов. Часто пользователи не используют шаблоны, а просто открывают подходящий рисунок и сохраняют его под новым именем.

При работе в нашей системе шаблоны не будут играть особой роли. Мы предусмотрим программную настройку всех параметров рисунка с учетом его масштаба, единиц и настраиваемых значений по умолчанию. Пользователи, при желании, смогут использовать шаблоны, в том числе создавать собственные. Создание собственных шаблонов будет проще, т. к. ручной настройки уже не потребуется.

Стандарты

Большую роль в проектировании (не только автоматизированном) играют стандарты. Строительное черчение производится в соответствии со стандартами Единой системы проектной документации для строительства (СПДС). Стандарты СПДС устанавливают, как должен выглядеть чертеж на бумаге. К сожалению, СПДС, в силу исторических причин, не учитывает особенностей автоматизированного проектирования. Современных стандартов на автоматизированное проектирование не существует вообще (если не считать ГОСТ 2.004-88, регламентирующего, какими символами выводить основные надписи на АЦПУ). А такие стандарты очень нужны. Прекрасно выполненные реалистичные модели зданий, оборудования и трубопроводов в лучшем случае могут использоваться как дополнительные иллюстративные материалы, а могли бы быть (разумеется, при дополнении отметками, размерами, надписями) и основной продукцией, в том числе передаваемой в электронном виде.

Оставим эту интересную тему для других изданий и разберемся, как мы в своей системе будем учитывать требования стандартов. Очевидно, что в такой большой системе, которую мы задумали, должна поддерживаться определенная политика стандартизации. Разделим вопрос на два логических направления:

- Как добиться соответствия чертежей на бумаге стандартам СПДС и надо ли это делать всегда?
- Как стандартизировать внутренние приемы работы с учетом специфики AutoCAD?

О соответствии чертежей на бумаге стандартам СПДС

Соответствие чертежей стандартам проверяется специальной службой нормоконтроля. Нормоконтроль в строительном проектировании был введен в 1982 году и во многом формально. Во многих проектных организациях требования о создании службы нормоконтроля просто игнорировали, в других чисто формально включили в список подписей графу *Н.контр*, а специальных людей, "натасканных" на эту работу, не выделили. В результате многие строительные чертежи (конечно, не все), выглядят так, будто надписи на них делала "курица лапой", шрифтом (скорее почерком), отдаленно напоминающим чертежный, а линии проведены "как бык ... хвостом провел".

Появление компьютерных чертежей, в которых трудно (но все-таки можно) добиться привычной "корявости", всегда воспринималось с восторгом. В строительстве нет, например, проблем с выяснением, какой шрифт в AutoCAD более всего соответствует ЕСКД, хотя такой вопрос постоянно возникает у конструкторов-машиностроителей, третируемых "злобствующими" нормоконтролерами.

Рассмотрев конфликты с СПДС, возникающие при автоматизированном черчении, можно выделить следующие вопросы:

- соответствие размеров стандартных форматов и размеров листов бумаги или границ печати на импортных принтерах и плоттерах;
- вид основной надписи, очень неудобно "врезающейся" в полезное поле листа;
- начертание и размер надписей;
- ширина линий;
- противоречия между требуемыми или допускаемыми СПДС упрощениями в строительных чертежах и возможностями САПР.

Форматы и размеры листа

Стандартные форматы часто не вписываются в форматы устройств печати. Например, в уходящую эпоху матричных принтеров, у которых при номинальном формате А3 фактическая ширина зоны печати была примерно 343 мм, вывод чертежа в формате А3 (ширина 420 мм) при расположении поперек рулона или А2 (высота 420 мм) при расположении вдоль рулона был невозможен. Некоторые проектировщики даже занимались приклеиванием недостающих полосок. Выход же в том, чтобы "узаконить" на уровне стандарта предприятия нестандартные форматы, например 340×841 или использовать опцию "вписать" при выводе чертежа на бумагу из системы AutoCAD. В последнем случае чертеж на бумаге получится не в масштабе, но это фактически допускается СПДС, т. к. в строительных чертежах (за исключением чертежей изделий и случаев, оговоренных соответствующими стандартами) масштабы не проставляют.

Это требование означает, что все необходимые размеры должны проставляться на чертеже, а производитель работ не должен производить никаких измерений с листа "по масштабу". В соответствии с этим допущением тиражируется, например, типовая проектная документация, в которой часто форматы уменьшаются так, чтобы образовались дополнительные технологические поля. Многие проектные организации вообще выдают заказчику чертежи, вписанные в меньший формат (например, чертеж формата А3, вписанный в бумажный лист А4 при печати на лазерном принтере, прекрасно читается и удобен в работе на стройке).

В то же время некоторые чертежи, такие как топографические планы и основанные на них генеральные планы и планы коммуникаций, должны выполняться именно в масштабе, т. к. там без измерений на бумаге не обойтись.

Практический вывод для нашей системы

Необходимо предусмотреть возможность вычерчивания форматов любых размеров, в том числе нестандартных. Решение о применении тех или иных размеров форматов должен принимать пользователь с учетом проводимой в проектной организации технической политики.

Вид основной надписи

Основная надпись (штамп) чертежа по форме 1 размером 55×185 мм занимает очень много места. Особенно это актуально для чертежей формата А3, очень популярного в России. Зачастую в чертеже остается недостаточно места для размещения основного изображения. В старое доброе время, до СПДС, в каждой организации существовали свои "стандарты" оформления листов. Во многих случаях "штампы" располагались вдоль границ формата. Такой же подход наблюдается во многих "Title Block", поставляемых с шаблонами чертежей в составе системы AutoCAD. "Узкие" штампы, размещенные вдоль нижней границы формата, были узаконены для некоторых видов документации (например, для жилищного строительства) и до сих пор применяются во многих проектных организациях.

Практический вывод для нашей системы

Необходимо предусмотреть возможность вычерчивания не только стандартных, но и компактных основных надписей. Решение о применении должен принимать пользователь с учетом проводимой в проектной организации технической политики.

Начертание и размер надписей

В строительных чертежах всегда имеется много надписей, включая размеры, причем в условиях дефицита места для их размещения. Размер шрифта по СПДС может колебаться от 2,5 до 7 мм в зависимости от назначения надписи. Наш многолетний опыт подсказывает, что от такого разнообразия следует отказаться и все надписи делать шрифтом высотой 2,5 мм на бумаге, за исключением заголовков, для которых рационально установить высоту 5 мм. Шрифты таких размеров хорошо читаются на "вписанных" чертежах и занимают минимальное место. Программное начертание большинства текстов мы будем делать именно такой высотой, разумеется, не лишая пользователя возможности выполнять отдельные надписи "по вкусу".

Все тексты мы будем писать стилем RU_CAD, устанавливая для него по умолчанию файл шрифта rucad.shx. В настройках системы будет предусмотрена возможность изменить шрифт по умолчанию на любой другой.

Отдельные надписи специального назначения (например, топографические), для которых установлены иные начертания и размеры, будут выполняться специальными стилями.

Ширина линий

Ширина линий (толщина по традиционной терминологии) в строительных чертежах, на наш взгляд, излишне детализирована в зависимости от масштаба и назначения линий. Общее количество "типоразмеров" ширины линий по стандарту более 50!

Но, как показывает практика, все можно свести к двум вариантам: тонкая и основная ("жирная").

В ruCAD "тонкой" будет считаться любая линия, которая всегда должна быть тонкой, и на бумаге, и на экране. Такие линии (любые примитивы) всегда будут создаваться с нулевым весом, а для полилиний будет еще и устанавливаться нулевая физическая ширина всех сегментов.

Линии, которые *могут* по желанию отображаться "жирными", должны создаваться с заданным весом. При использовании полилиний это означает задание ширины 0, а веса — отличного от нуля. Возможность выбора веса линий должна предоставляться соответствующими программами.

Линии, которые *всегда* должны быть определенной ширины, следует рисовать полилиниями с заданной физической шириной, соответствующей требуемой ширине на бумаге при заданном масштабе. Вес таких полилиний необходимо устанавливать соответствующим требуемой ширине линий на бумаге, для того чтобы при операциях расчленения (т. е. превращения полилинии в отрезки и дуги) и изменения физической ширины полилинии сохранялся требуемый облик объекта.

Реализация программной поддержки ширины линий не очень сложна, но требует определенной методичности. Каждая программа должна знать, какие объекты и в каких случаях могут или должны рисоваться тонкими или "жирными" линиями, и должна предоставлять пользователю возможность выбора, в том числе и нестандартной ширины.

Практический вывод для нашей системы

Предусматриваем сокращенные, по сравнению с СПДС, размерные ряды высоты текста и ширины линий с предоставлением пользователям возможности выбора и корректировки значений по умолчанию.

Избыточная детализация

Стандарты СПДС, помимо унификации, были направлены на ускорение процесса черчения за счет различных упрощений и применения схематичных изображений. Например, на разрезах схематично показывались ограждения лестничных маршей или контуры стропильных ферм, в чертежах санитарно-технических систем упрощенно изображались контуры оборудования и трубопроводов. Стремление избежать излишней детализации было оправдано повышением производительности труда, но, в некоторых случаях, приводило к тому, что на стройке приходилось выполнять работы "по соображению".

Положение усугублялось тем, что в строительном проектировании применялось множество типовых узлов и изделий, рабочие чертежи которых подрядная организация должна была "выписывать" где-нибудь в Тбилисском ЦИТП. Далеко не все "серии" имелись даже в производственно-технических отделах подрядных организаций, и прорабы постоянно клянчили их у проектировщиков, иногда только для того, чтобы убедиться, что таинственную *Опору А14Б435.000-07* они изготавливали уже 238 раз, только не знали, что она теперь так называется.

В условиях постсоветского времени проектирование и строительство осуществляется не только для единственного заказчика в лице государства. Пример недостаточной детализации чертежей и спецификаций мы уже приводили в *главе 2*. Помимо прочего, проектные организации работают теперь в условиях жесткой конкуренции. Если потенциальный заказчик, знакомясь с проектами "конкурирующих фирм" в одной увидит стандартную схему лестницы и скучные типовые фасады, а в другой с любовью и детально прорисованные перила и балясины (да еще индивидуальной конструкции, да еще именно такие, какие ему снятся), множество интересных деталей

на фасадах, наподобие уникальных флюгеров или водосточных труб, выбор его можно предугадать.

При использовании САПР достичь любой детализации несложно безо всякого увеличения времени на проектирование, а эффектность (и эффективность) может быть очень велика. В том числе за счет совмещения рабочего и монтажного проектирования, о которых уже говорилось в *главе 2*.

Практический вывод для нашей системы

Необходимо предусмотреть возможность вычерчивания с избыточной, по сравнению с СПДС, детализацией, а пользователь должен иметь возможность выбирать, как он будет рисовать узлы и детали.

Стандартизация приемов работы с AutoCAD

Тема эта, как показывают многочисленные дискуссии на форумах в Интернете, очень актуальна. При коллективной работе стандартизация, безусловно, необходима. В последних версиях AutoCAD, хоть и в зачаточном виде, предусмотрена возможность стандартизации решений. Пока она сводится к установке наборов допустимых свойств именованных объектов, таких как слои, текстовые стили, типы линий, размерные стили.

Описанный стандарт сохраняется в DWS-файле, аналогично шаблону. Файл стандартов может быть назначен нескольким файлам рисунков. Рисунок AutoCAD, связанный с файлом стандартов, можно с помощью команды CHECKSTANDARDS периодически проверять на соответствие этим стандартам.

Это уже лучше, чем ничего, но все еще очень мало. Стандартизация должна охватывать более широкий круг вопросов. Примером стандартизации могут быть требования "Indiana University CAD deliverable requirements for IU projects", с которыми можно ознакомиться в документе **http://www.indiana.edu/~uao/IU_as-built_cad_requirements.pdf** (и других, ссылки на которые можно найти на сайте **http://www.indiana.edu/~uao**). Полезно также познакомиться с сайтом **http://www.nationalcadstandard.org**.

Конечно, эти стандарты мало подходят для российских условий, но они интересны в плане методического подхода и конкретных регламентов.

Наша система предназначена для коллективной работы, будет использоваться в смежных разделах и, естественно, она должна поддерживать определенную стандартизацию методов работы и конечных результатов. Пока не выработан какой-то "Российский стандарт работы с AutoCAD" мы попробуем, с учетом мнений, высказывавшихся на форумах, выработать и поддержать программными решениями собственный "стандарт ruCAD".

Опуская для экономии драгоценного места общие фразы, приведем в сжатом виде основные положения, которыми будем руководствоваться при программной реализации.

Примечание

Разъяснения о том, почему мы рекомендуем тот или иной прием, будут приведены в комментариях.

Стандарт приемов работы ruCAD для строительного проектирования

Основные термины

- **Рисунок** — изображение на экране компьютера, сохраняемое в DWG-файле.
- **Законченный рисунок** — DWG-файл, предназначенный для совместного использования. Все требования стандарта распространяются на законченные рисунки. В незаконченных рисунках, находящихся в стадии разработки, допускаются отступления от настоящего стандарта.
- **Чертеж** — рисунок, вычерченный на бумажном носителе в определенном масштабе.
- **Масштабируемые символы** — тексты, элементы размеров, таблицы, форматы, условные знаки и другие изображения, размеры которых устанавливаются государственными стандартами для бумажного носителя.
- **Реальные объекты** — здания, сооружения, изделия и прочие предметы, изображаемые в рисунке.
- **Масштаб чертежа** — масштаб, в котором чертеж будет выведен на бумажный носитель.
- **Единица измерения рисунка** — наименование единицы измерения, соответствующей условной линейной единице рисунка в AutoCAD.
- **Рабочая группа** — пользователи системы AutoCAD, в том числе из различных организаций, использующие единый стандарт работы. Присоединение к рабочей группе может быть добровольным или, в рамках одной организации или нескольких организаций (по взаимному соглашению), обязательным.

Единицы рисунка

За единицу измерения рисунка в *пространстве модели* принимается 1 миллиметр для чертежей зданий и изделий и 1 метр для чертежей топографической подосновы, генерального плана и коммуникаций.

Код единицы рисунка в *пространстве модели* сохраняется в системной переменной INSUNITS (4 — для миллиметров, 6 — для метров).

Наименование единицы рисунка в *пространстве модели* отображается в окне **Drawing Properties** (Свойства рисунка) на вкладке **Custom** (Прочие) во второй строке.

За единицу измерения рисунка в *пространстве листа* принимается 1 миллиметр чертежа.

Комментарий

В AutoCAD нет жестко установленной системы единиц (метры, миллиметры и т. п.). В одном рисунке единица может соответствовать одному миллиметру, в другом — одному метру, в третьем — одному километру. Система ruCAD предназначена для строительного проектирования, в котором за единицы принимаются только миллиметры и метры (для генпланов и сетей). Наименование единицы будет устанавливать пользователь при создании рисунка, и оно будет сохраняться в файле рисунка в словаре свойств. Системная переменная INSUNITS самое подходящее место для хранения кода единицы, т. к. позволит одновременно использовать возможность вставки рисунка в виде блока через Центр управления штатными методами системы AutoCAD.

Преобразование единиц рисунка при необходимости совмещения, например, при вставке на генеральный план, выполненный "в метрах", плана здания, выполненного "в миллиметрах", мы будем осуществлять программным путем. Пользователь не должен вникать в такие детали, но он (или другой пользователь, не использующий ruCAD), сможет воспользоваться возможностями Центра управления.

За тип линейных единиц рисунка принимаются десятичные (Decimal, значение системной переменной LUNITS = 2). Точность отображения (Precision) линейных единиц принимается равной 1 для рисунков с наименованием единицы "мм" и 2 для наименования единицы "метр" (системная переменная LUPREC).

Комментарий

Формат единиц в AutoCAD может иметь разный вид. Мы будем использовать только десятичный вид. Так называемый "архитектурный" формат к российской архитектуре отношения не имеет (хотя в древних проектах иногда для кладочных планов применялась единица "кирпич").

Точность измерений можно устанавливать любую, но, руководствуясь здравым смыслом, мы будем автоматически устанавливать точность 1 для рисунков с единицей "миллиметр" и 0.001 для рисунков с единицей "метр".

За тип угловых единиц рисунка принимаются Градусы/Минуты/Секунды (Deg/Min/Sec, значение системной переменной AUNITS = 2). Точность отображения (Precision) угловых единиц принимается равной 4 (системная переменная AUPREC).

Масштабы

Реальные объекты изображаются в пространстве модели в натуральную величину (масштаб 1:1) в принятых единицах рисунка. *Масштабируемые символы* изображаются в пространстве модели в таком масштабе, при котором их размеры в чертеже будут соответствовать государственным стандартам, а при изображении в пространстве листа в масштабе 1:1 в миллиметрах.

Комментарий

Реальные объекты в *пространстве модели* должны изображаться в натуральную величину в принятых единицах рисунка! Это фундаментальный принцип системы AutoCAD, который не должен бы разжевываться в книге, рассчитанной на разработчиков, т. е., как минимум, не начинающих пользователей. Подчеркивать это мы вынуждены только потому, что встречаются программы и целые "системы", авторы которых этого в свое время не усвоили, а теперь "калечат умы" своих наивных пользователей.

Дополнительные разъяснения для "упорствующих в ереси" мы еще раз приведем далее в этой главе.

Изображение масштабируемых (иногда называемых внемасштабными) символов должно осуществляться пропорционально заданному масштабу рисунка программным путем без дополнительных усилий пользователя.

Пространства листа и модели

В пространстве модели изображаются реальные объекты и масштабируемые символы, связанные местоположением с реальными объектами.

В пространстве листа изображаются форматы, заголовки изображений, тексты примечаний и указаний и другие элементы оформления, не требующие постоянной привязки к реальным объектам, изображенным в пространстве модели. Рекоменду-

ется размещение в пространстве листа размеров, ассоциированных с примитивами, изображенными в пространстве модели.

Масштаб фрагментов пространства модели (включая масштабируемые символы, изображенные в пространстве модели), отображаемых в пространстве листа в законченном рисунке должен соответствовать масштабу соответствующей вкладки (Layout).

Комментарий

Рекомендации о разделении изображений по пространствам модели и листа не являются столь категоричными, как требование рисования в натуральную величину. Можно рисовать вообще все (в том числе элементы оформления) в пространстве модели. Это часто делают пользователи, задержавшиеся в развитии на уровне AutoCAD R10. Большого вреда для окружающих от этого нет, такие пользователи просто усложняют работу себе и своим коллегам, вынужденным работать с их материалами, не пользуясь замечательными возможностями системы AutoCAD. По этому вопросу мы также дадим дополнительные комментарии.

Системы координат и их описание

В законченном рисунке с началом *мировой системы координат* (МСК) должно совпадать характерное место изображения в пространстве модели (пересечение осей, угол здания и т. п.). Это место должно быть единым для всех изображений одного реального объекта, например для планов всех этажей. Для топографических планов и основанных на них рисунках начало мировой системы координат совпадает с местной системой координат населенного пункта.

Комментарий

Не следует принимать за начало координат что-нибудь неопределенное или "подходящее", например угол формата. В рамках одного проекта все рисунки, для которых рано или поздно понадобится совмещение, должны иметь одно начало МСК. В этом случае точка вставки чаще всего будет 0, 0.

Если в пространстве модели изображается формат, то для позиционирования изображений в рамке должен переноситься или поворачиваться формат, а не основное изображение.

Комментарий

Забудьте про "бумажное" проектирование, при котором часто передвигают по поверхности формата наклейки и темплеты. Двигайте рамку, а лучше видовой экран, а не модель, иначе у вас изменится точка вставки, и рисунок будет малопригоден для совмещения с другими.

Точка вставки

В законченных рисунках точка вставки должна совпадать с началом мировой системы координат. В свойствах рисунка должно сохраняться словесное описание точки вставки, облегчающее работу при вставке рисунка в виде блока.

Комментарий

Для того чтобы не передвигать несколько раз вставленный блок, удобно, если в момент вставки выводится не стандартное приглашение "Точка вставки:", а более осмысленное "Точка вставки (пересечение осей 1—А):". Для реализации этого необходимо запоминать словесное описание точки вставки. Удобнее всего это сделать на вкладке **Custom** (Прочие) диалогового окна **Drawing Properties** (Свойства рисунка).

Направления осей

Направление нуля градусов оси X мировой системы координат в системе AutoCAD принимается на восток. Для рисунков, использующих топографическую подоснову, в которых наименование геодезической оси X, обычно (но не всегда) направленной на север, может не совпадать с наименованием оси AutoCAD, направленной на север (обычно Y), в свойствах рисунка (Drawing Properties) должно сохраняться направление оси X.

Размеры

Все размеры выполняются ассоциативными с использованием средств объектной привязки при создании размера. Рекомендуется простановка размеров общего назначения в пространстве модели, специальных размеров — в пространстве листа.

Комментарий

О возможных вариантах простановки размеров и влиянии этого на программную реализацию будет дан дополнительный комментарий.

Точность округления размеров принимается 1 для рисунков с наименованием единицы миллиметр и 0.001 для наименования единицы метр.

Все объекты должны рисоваться с максимальной точностью, с использованием ручного ввода размеров, объектной и шаговой привязки.

Свойства объектов

Все объекты должны создаваться с цветом ByLayer (ПОСЛОЮ), за исключением таких, для которых государственными стандартами установлены конкретные цвета.

Комментарий

Излишнее обилие цветов отдельных примитивов — типичный признак малоопытного пользователя. Для улучшения восприятия рисунка достаточно изменять цвет слоя. Обоснованное применение различных цветов — назначение для каждого цвета ширины пера при выводе на печать. Но и в этом случае достаточно двух, в самом крайнем случае четырех цветов. Ширину линий мы будем устанавливать программным путем, это позволит избежать излишних хлопот с цветом.

Все надписи (тексты, атрибуты блоков, тексты размеров) должны создаваться с текстовым стилем RU_CAD, за исключением специальных (топографических и иных) надписей, для которых начертание шрифта устанавливается иными стандартами.

Слои

Распределение примитивов по слоям в *законченном* рисунке производится по следующим принципам:

- на слое 0 не должно быть никаких примитивов;
- на специальных слоях (DEFPOINTS, URLLAYER и т. п.), создаваемых штатными средствами системы AutoCAD, должны размещаться только примитивы, созданные этими средствами;
- группировка примитивов по слоям производится по логическому принципу принадлежности к виду изображаемых объектов, а не по свойствам примитивов;

- имена слоев формируются по схеме:

  ПРЕДОК_ОБЪЕКТ_ВАРИАНТ_ВАРИАНТ, где:

  - ПРЕДОК — имя слоя, от которого "произошел" данный слой (возможно, включающее "имя_дедушки";
  - ОБЪЕКТ — имя, характеризующее назначение объектов, изображенных на слое;
  - ВАРИАНТ — дополнительные характеристики (этаж, стадия проектирования и т. п.).

При программной реализации рекомендуется использование "конструкторов имен слоев", обеспечивающих единообразное именование. Примеры имен слоев приведены в табл. 3.2.

Таблица 3.2. *Пример некоторых имен слоев*

Имя слоя	Объекты на слое
АС	Архитектурно-строительная часть
АС_ОСИ	Архитектурно-строительная часть. Оси
АС_ТАБЛИЦЫ	Архитектурно-строительная часть. Таблицы
АС_ТЕКСТЫ	Архитектурно-строительная часть. Тексты
АС_ТЕКСТЫ_ОБЩИЕ	Архитектурно-строительная часть. Тексты для всех разделов
АС_ТЕКСТЫ_СПЕЦ	Архитектурно-строительная часть. Тексты специальные
АС_РАЗМЕРЫ	Архитектурно-строительная часть. Размеры
АС_РАЗМЕРЫ_ОБЩИЕ	Архитектурно-строительная часть. Размеры для всех разделов
АС_РАЗМЕРЫ_ОБЩИЕ_1ЭТ_РД	Архитектурно-строительная часть. Размеры для всех разделов. 1-й этаж рабочей документации
АС_РАЗМЕРЫ_200_1ЭТ_РД	Архитектурно-строительная часть. Размеры для фрагментов масштаба 1:200. 1-й этаж рабочей документации
АС_РАЗМЕРЫ_50_1ЭТ_РД	Архитектурно-строительная часть. Размеры для фрагментов масштаба 1:50. 1-й этаж рабочей документации
АС_РАЗМЕРЫ_СПЕЦ_2ЭТ_РД	Архитектурно-строительная часть. Размеры специальные. 2-й этаж рабочей документации
АС_ОТМ_ОБЩИЕ	Архитектурно-строительная часть. Отметки для всех разделов
АС_ОТМ_СПЕЦ	Архитектурно-строительная часть. Отметки специальные
АС_СТЕНЫ	Архитектурно-строительная часть. Стены
АС_СТЕНЫ_ПРОЕКТ	Архитектурно-строительная часть. Стены проектируемые

Таблица 3.2 *(окончание)*

Имя слоя	Объекты на слое
АС_СТЕНЫ_ПРОЕКТ_1ЭТ_ТЭО	Архитектурно-строительная часть. Стены проектируемые. 1-й этаж. Стадия ТЭО
АС_СТЕНЫ_ПРОЕКТ_1ЭТ_ТП	Архитектурно-строительная часть. Стены проектируемые. 1-й этаж. Стадия ПРОЕКТ
АС_СТЕНЫ_ПРОЕКТ_1ЭТ_РД	Архитектурно-строительная часть. Стены проектируемые. 1-й этаж. Стадия РД
АС_СТЕНЫ_ПРОЕКТ_ОТВ	Архитектурно-строительная часть. Стены проектируемые. Отверстия
АС_СТЕНЫ_ПРОЕКТ_ОТВ_1ЭТ_РД	Архитектурно-строительная часть. Стены проектируемые. Отверстия. 1-й этаж. Стадия РД
АС_СТЕНЫ_ПРОЕКТ_ОТВ_ОВ_1ЭТ_РД	Архитектурно-строительная часть. Стены проектируемые. Отверстия для ОВ. 1-й этаж. Стадия РД
АС_СТЕНЫ_ПРОЕКТ_ОТВ_ВК_1ЭТ_РД	Архитектурно-строительная часть. Стены проектируемые. Отверстия для ВК. 1-й этаж. Стадия РД
АС_СТЕНЫ_СУЩ	Архитектурно-строительная часть. Стены существующие
АС_СТЕНЫ_СУЩ_ОБМЕР_1953	Архитектурно-строительная часть. Стены существующие. Обмеры 1953 года
АС_СТЕНЫ_СУЩ_ОБМЕР_2002	Архитектурно-строительная часть. Стены существующие. Обмеры 2002 года
ОВ	Отопление и вентиляция
ОВ_ЗД	Отопление и вентиляция. Задания смежникам
ОВ_ЗД_СТЕНЫ_ОТВ_АР_1ЭТ_РД	Отопление и вентиляция. Задания смежникам. Отверстия в стенах 1 этажа для АР. Стадия РД

При формировании имени слоя рекомендуется в качестве разделителя слогов использовать символ подчеркивания. Длину имени слоя рекомендуется принимать не более 31 символа.

Комментарий

Короткое имя слоя будет полностью помещаться в полях для редактирования и выбора, облегчит использование рисунков в ранних версиях AutoCAD, сохранение слоев в виде файлов и вставку файлов-слоев в виде блоков и внешних ссылок.

Для обеспечения единообразия свойств слоев в рабочих группах ведется классификатор слоев. Формат и способы хранения классификатора слоев могут быть различными в зависимости от используемых программных средств, количества членов рабочей группы и технических возможностей. Рекомендуемые способы хранения классификатора слоев (в предпочтительном порядке):

- база данных на удаленном SQL-сервере (Interbase, MSSQL) с постоянным доступом для чтения всех членов рабочей группы;
- использование клиентских наборов данных на локальных компьютерах, периодически синхронизируемых с базой данных на удаленном SQL-сервере;

- базы данных на локальных компьютерах в формате Microsoft Access, периодически синхронизируемые в пределах рабочей группы;
- файлы на локальных компьютерах в формате XML, периодически синхронизируемые в пределах рабочей группы.

Программный доступ к классификатору слоев из Visual LISP должен предусматриваться с использованием технологий COM и ADO.

В рабочей группе для обеспечения единообразия имен слоев допускается использование файлов стандартов в формате, поддерживаемом системой AutoCAD.

Каждый классифицированный слой должен иметь следующие параметры:

- основное имя слоя длиной до 31 символа;
- альтернативное имя слоя (английское); изменение основных имен на альтернативные (при необходимости) должно выполняться программными методами;
- комментарий (описание) длиной до 250 символов на "нужном" языке;
- уникальный 8-символьный код слоя, использующийся в качестве имени файла при записи слоя на диск в операционных системах, не гарантирующих поддержку и целостность имен файлов, включающих полное имя слоя;
- цвет на экране по умолчанию;
- цвет на бумаге;
- тип линии по умолчанию;
- вес линии;
- стиль печати;
- возможность печати слоя.

Допускается введение других необязательных свойств (метод создания объектов, видимость при различных масштабах и т. п.). Для улучшения визуального отображения слоев в виде иерархического дерева рекомендуется использовать целочисленные коды слоя (уникальный) и слоя-предка.

Параметры слоев сохраняются в базе данных. Допускается в базе данных хранить только имя слоя и ссылку на INI-файл, в котором в секциях, соответствующих именам слоев, хранятся остальные параметры слоя.

Блоки

Все примитивы, входящие в состав блоков, должны создаваться на слое 0 с цветом ByBlock (ПОБЛОКУ). Исключением являются блоки, изображающие предметы, для которых иными стандартами установлен конкретный цвет, например дорожные знаки.

Отрисовка примитивов и установка точки вставки блоков производятся с максимальной точностью при помощи средств шаговой и объектной привязки, а также с использованием клавиатурного ввода координат, углов и расстояний.

Точка вставки блоков назначается в характерных узлах, соответствующих назначению блока. Для блоков условных знаков, у которых государственными стандартами нормируются координаты размещения условного знака на чертеже, точки вставки должны соответствовать этим местам.

Именованные объекты

Другие именованные объекты (блоки, стили текста и размеров, типы линий, вкладки, виды, системы координат и т. п.) именуются по следующим принципам:

- объектам, видимым пользователю, предпочтительно назначаются русские имена;
- объектам, имена которых не обязательно явно выбирать пользователю, назначаются английские имена;
- в зависимости от характера работ и квалификации пользователей допускается применение только русских или только английских имен. При назначении английских имен должны использоваться английские слова и их сокращения, а не транслитерация русских слов (WALL, WINDOW, а не STENA, OKNO).

Подготовка к архивированию

Все неиспользуемые объекты перед архивированием или передачей рисунка должны удаляться командой PURGE.

Распределение изображений по файлам

Законченный рисунок в пространстве модели должен содержать изображение только одной модели (плана, разреза и т. п.). В пространстве листа законченного рисунка на различных вкладках могут находиться изображения различных частей одной модели (сводный план и фрагменты плана, разрез и увеличенные узлы этого разреза).

Изображения, носящие самостоятельный характер и не являющиеся частями единой модели (типовые узлы, другие проекции), должны размещаться в отдельных файлах.

При необходимости допускается создавать сборный рисунок, в который в виде внешних ссылок вставляются другие файлы проекта.

Имена файлов

Имена файлов рисунков должны соответствовать стандарту предприятия. Рекомендуется формировать имена рисунков из обозначения документа, записываемого в основную надпись. В именах файлов переменная часть должна находиться в начале имени.

Это необходимо для более наглядного отображения имен файлов в диалоговых окнах (рис. 3.1 и 3.2).

Система каталогов

Файлы рисунков хранятся на рабочей станции исполнителя и в электронном архиве. Структура папок в электронном архиве должна соответствовать принятой на предприятии системе документооборота. На рабочей станции исполнителя структура папок должна соответствовать структуре электронного архива, а также структуре обозначения документов.

Замечание

Мы не имеем в виду, что на рабочей станции хранится копия электронного архива! Структура папок в архиве должна соответствовать пользовательской, только "глубже" отведенного для пользователя каталога. Например, если пользователю задано имя его верхнего каталога WATER, то он на своем компьютере может создавать подкаталоги WATER\PLANS и WATER\DETAILS, а при передаче документов в архив аналогичные каталоги должны создаваться и там.

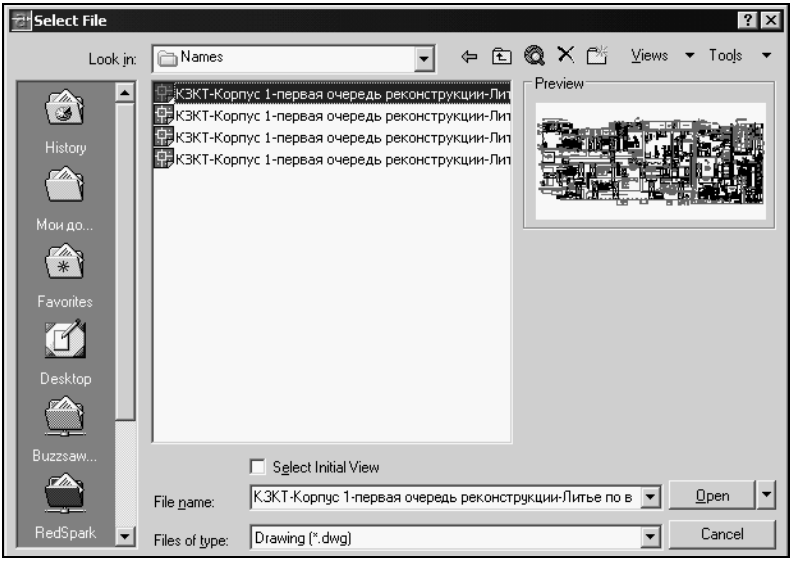


Рис. 3.1. Постоянная часть в начале имени файла

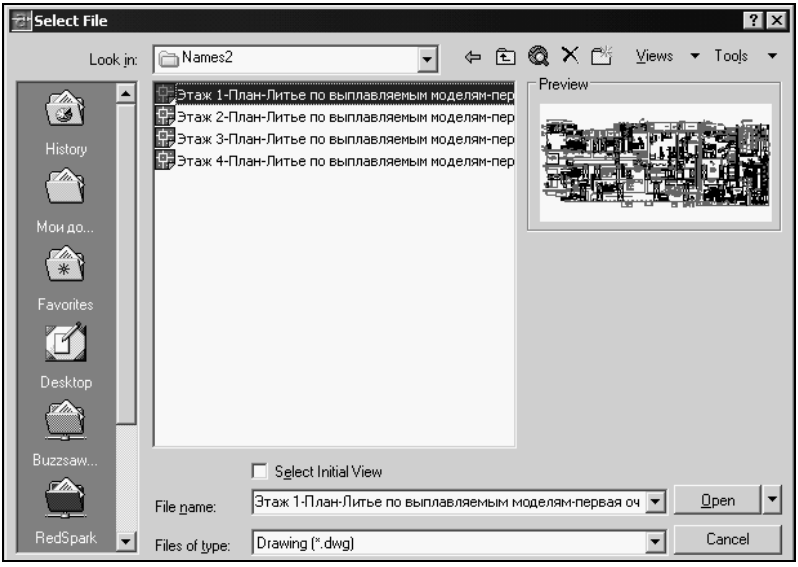


Рис. 3.2. Переменная часть в начале имени файла

В проектной организации может действовать следующая структура обозначений документов:

- каждое предприятие-заказчик имеет свой код ЗАК;
- каждый договор с заказчиком (соответствующий термину "стройка") имеет свой код ДОГ;

- на предприятии может быть несколько площадок или отдельных внеплощадочных объектов, имеющих собственный код ПЛОЩ;
- на каждой площадке может быть несколько зданий, каждое из которых имеет свой код ОБ (обычно номер по генплану). Кроме зданий, на площадке могут быть общеплощадочные объекты и инженерные сети, каждая из которых также может иметь свой код ОБ;
- по каждому объекту в разные периоды или одновременно может выполняться несколько видов проектов (очереди строительства, реконструкция, техперевооружение и т. п.). Каждый вид проекта имеет свой код РАБ;
- любой проект может выполняться в одну или несколько стадий, каждая из которых имеет код СТ;
- по каждой работе может выполняться несколько разделов (основных комплектов) документации, каждый из которых имеет код МАРКА;
- все коды имеют длину от 1 до 4 символов (предпочтительно "осмысленного" вида). Коды выбираются из базы данных, ведущейся в проектной организации.

Обозначение любого комплекта документов должно быть уникальным. В основной надписи рабочих чертежей обозначение имеет примерно такой вид:

ЗАК.ДОГ.ПЛОЩ.ОБ.РАБ.СТ-МАРКА

К обозначению комплекта добавляется номер листа. Кроме того, в конец обозначения могут дописываться добавочные обозначения, установленные стандартами для некоторых документов (общие виды, спецификации оборудования и т. п.).

При необходимости в обозначение комплекта могут вводиться дополнительные коды или исключаться отсутствующие. Допускается замена отсутствующих кодов нулями для сохранения постоянной структуры и длины обозначения. Структура обозначений разрабатывается при непременном участии главного инженера (главного архитектора) проекта.

Примеры обозначений:

- 238.546-01.ПЛ2.000.000.Р-ГП — рабочие чертежи генплана площадки 2 завода автоприцепов, разрабатываемые по договору 546-01.
- 238.546-02.ПЛ2.012.1ОЧ.Р-АР — рабочие чертежи марки АР первой очереди строительства корпуса 12 на площадке 2 завода автоприцепов, разрабатываемые по договору 546-02.
- 238.547.КПД108.ЦТП.Р-ТМ — рабочие чертежи марки ТМ центрального теплового пункта для крупнопанельного жилого дома № 108 завода автоприцепов, разрабатываемые по договору 547.

При формировании структуры папок для каждого кода создается папка с соответствующим именем. В каждой папке могут быть или только вложенные папки, или только файлы.

Полные имена файлов для приведенных выше примеров будут иметь вид:

- D:\DATA\238\546-01\ПЛ2\000\000\Р\ГП\01_05-ГП_Р_000_000_ПЛ2_546-01_238.DWG — файл генплана, в котором, на разных вкладках находятся отпечатанные листы 1—5;

- D:\DATA\238\546-02\ПЛ2\012\1ОЧ\Р\АР\01-АР_Р_1ОЧ_012_ПЛ2_546-02_238.DWG — файл строительной части, в котором находится отпечатанный лист 1;
- D:\DATA\238\546-02\ПЛ2\012\1ОЧ\Р\АР\02_04-АР_Р_1ОЧ_012_ПЛ2_546-02_238.DWG — файл строительной части, в котором на разных вкладках находятся отпечатанные листы 2—4;
- D:\DATA\238\547\КПД108\ЦТП\ТМ\01-ТМ_ЦТП_КПД108_547_238-01.DWG — файл тепломеханической части, в котором находится отпечатанный лист 1;
- D:\DATA\238\547\КПД108\ЦТП\ТМ\01-ТМ_ЦТП_КПД108_547_238.DWG — файл тепломеханической части, в котором находится отпечатанный лист 2.

Замечание

Достаточно длинные имена файлов, частично дублирующие полный путь, нужны для того, чтобы избежать коллизий при вставке файлов с одинаковыми именами, но с разным содержанием, и предотвратить путаницу при "выпадании" файлов с короткими именами из своих папок.

Свойства рисунка

Для организации системы документооборота или хотя бы наведения элементарного порядка в файловой системе необходимо иметь визуальный и программный доступ к свойствам файлов. В файловой системе NTFS в контекстном меню свойств документа для DWG-файлов, помимо свойств, общих для всех документов, отображаются и специфические свойства, устанавливаемые внутри системой AutoCAD посредствам диалогового окна **Drawing Properties** (Свойства рисунка). Имеется возможность осуществлять программный доступ к этим свойствам, в том числе на компьютерах без установленной системы AutoCAD.

Рекомендуется следующий порядок использования свойств рисунка.

- Вкладка **Summary** (Документ):
  - **Title** (Название) — наименование рисунка, например: *"Водомерный узел № 2. Вид А"*.
  - **Subject** (Тема) — наименование стройки, объекта, например: *"КЗКТ. Корпус 15. 3-я очередь"*.
  - **Author** (Автор) — автор рисунка, например *"Типроавтоагрегат. ОТЭ. Исп. Зуев С. А. ГИП Корляков Н. Л."*.
  - **Keywords** (Ключевые слова) — ключевые слова для поиска, например: *"КЗКТ водоснабжение корпус 15"*.
  - **Comments** (Заметки) — комментарий к файлу, например: *"Дополнительный водомерный узел № 2 корпуса 15 КЗКТ. Вариант с обводной линией"*.
  - **Hyperlink base** (База гиперссылки) — для локальных файлов корневой каталог проекта, например *"http://195.54.28.21"*.
- Вкладка **Custom** (Прочие), на которой отображаются и устанавливаются 10 пользовательских имен свойств и их значений:
  - Имя свойства "Масштаб 1:" (код 300) — масштаб чертежа, например: *"25"*.
  - Имя свойства "Ед. изм" (код 301) — единица измерения рисунка, например: *"мм"*.

- Имя свойства "ТчкВст" (код 302) — словесное описание точки вставки, например: *"Пересечение оси А и пола на отм. 0.000"*.

Использование остальных пользовательских имен свойств и их значений (коды 303–309) допускается по усмотрению пользователя.

Комментарии к рисункам

Информации, записанной непосредственно в рисунке, может быть недостаточно. При работе в Windows 98 свойства рисунка нельзя просматривать средствами ОС. Информация о рисунке может понадобиться другим программам, не умеющим извлекать ее из DWG-файла. Для решения этих вопросов каждый рисунок может дополняться файлами ИМЯ_РИСУНКА.INI и ИМЯ_РИСУНКА.TODO.

Файл ИМЯ_РИСУНКА.INI имеет формат INI-файла. В него может записываться любая информация, в том числе дублирующая свойства рисунка (Drawing Properties).

Файл ИМЯ_РИСУНКА.TODO создается при необходимости ведения текущих заметок по текущему рисунку. Это обычный текстовый файл, в который информация записывается в произвольной форме. Рекомендуется в первой строке такого файла записывать сигнатуру .LOG, при наличии которой текстовый редактор Notepad автоматически вставляет в файл дату и время каждой записи.

Комментарии к файлам и папкам

Для облегчения работы с файловой системой в каждой папке может создаваться файл dirinfo.ini (листинг 3.1). В этот файл записываются комментарии к вложенным папкам и к файлам. Примечания к папкам записываются в секцию `[FOLDERS]`, примечания к файлам — в секцию `[FILES]`. Для каждого файла или каталога создается переменная с именем, равным имени каталога или файла, включая расширение, и со значением комментария к папке или файлу.

Листинг 3.1. Пример файла dirinfo.ini

```
[FOLDERS]
KZKT=Курганский завод колесных тягачей
IMZ=Ирбитский мотоциклетный завод
CHEL=Карта г. Челябинска
[FILES]
structure_01.dwg=Структура информационной системы
```

Системные переменные

Системные переменные USERR1, ..., USERR5, USERI1, ..., USERI5, USERS1, ..., USERS5 рекомендуется использовать только в тех случаях, когда возможно использование именно системной переменной AutoCAD, например, для управления доступом к меню средствами языка DIESEL или для обмена данными между программами, написанными на Visual LISP и VBA.

Резервируется для стандартных целей только системная переменная USERS5. В ней в виде строки длиной 250 символов должны храниться логические переменные доступа к различным привилегиям. Символ "0" в любой позиции строки соответствует запрещению, а любой другой символ — разрешению на привилегию с номером по-

зиции символа. **Первые 40 символов переменной USERS5 зарезервированы для системы ruCAD**.

Дополнительные комментарии к положениям стандарта

В этом разделе мы приводим дополнительные разъяснения к важнейшим требованиям стандарта работы с рисунком потому, что их неверное понимание разработчиком может привести к очень неприятным последствиям.

Масштаб рисунка

Замечание

Реальные объекты в *пространстве модели* должны изображаться в натуральную величину в принятых единицах рисунка!

Мы еще раз повторяем этот фундаментальный принцип AutoCAD.

При традиционном "бумажном" черчении исполнитель, в зависимости от размеров объекта и листа бумаги, сначала задается масштабом чертежа и затем, в процессе черчения, вычерчивает все объекты в принятом масштабе, непрерывно, возможно незаметно для себя, соображая:

— Так, надо нарисовать стену толщиной 640 мм, у меня масштаб 1:200, значит, провожу параллельные линии на расстоянии 3,2 мм...

В голове у человека как бы работает собственный компьютер, занятый исключительно пересчетом размеров. При масштабе 1:100 это незаметно, а при неудобных масштабах (например, при подрисовке на лист типового проекта, снятого с произвольным уменьшением до масштаба, скажем, 1:127) голова очень быстро заболит.

При работе с системой AutoCAD этого делать не надо, все рисование производится В НАТУРАЛЬНУЮ ВЕЛИЧИНУ, или, иначе говоря, в масштабе 1:1!

Так как все эти элементы "на ходу" изменить нельзя, величиной масштаба чертежа на бумаге необходимо задаться заранее, но этот масштаб будет применяться только для масштабируемых символов.

При натуральном масштабе размеры всех элементов задаются и показываются в натуральную величину для принятых единиц рисунка.

Например, если надо провести оси здания с шагом 6000 мм, то при единице рисунка "миллиметр" величина смещения осей задается 6000, а при единице рисунка "метр" — 6.0.

Само понятие масштаба применимо только к случаю вывода рисунка на бумагу. Вот там, на бумаге, мы и должны обеспечить нормативные размеры масштабируемых символов. В документации по системе AutoCAD приводятся примеры вычислений масштабных коэффициентов, применяемых для рисования масштабируемых символов. Пользователям нашей системы эти знания не потребуются — мы обеспечим полную автоматизацию этих процессов.

Фактический масштаб бумажной копии может быть:

- *намеченным*, т. е. таким, которым задались при начале работы и в соответствии с которым нарисованы все масштабируемые символы;
- *фактическим*, в случае, если чертеж вписывается в размер формата принтера или плоттера.

Для того чтобы чертеж соответствовал стандартам, следует стремиться к совпадению намеченного и фактического масштаба печати.

В случае, например, с печатью топографической подосновы для последующего "ручного" использования это требование является обязательным, а вот план здания можно без ущерба и "вписать" в лист, заботясь только о том, чтобы остались читаемыми надписи.

Управление шириной линий

Как мы уже отмечали в *главе 2*, при разработке чертежей нам потребуются линии различной ширины.

Примечание

Не следует применять термин "толщина линии", использующийся в традиционном черчении, к объектам, создаваемым в системе AutoCAD. Даже такие простые примитивы, как отрезки, в AutoCAD могут иметь *высоту* (thickness), иногда называемую "толщиной".

Одни и те же объекты в разных комплектах чертежей, или даже в одном комплекте, должны отображаться линиями разной ширины. Из-за этого возникало немало трудностей. Например, на строительном плане здания существующие стены рисовались тонкими линиями, а проектируемые — основными, на плане вентиляционной установки эти же стены все должны были быть показаны тонкими линиями (да еще без штриховок), а вентиляционное оборудование — основными. На электротехнических планах и стены, и вентоборудование (если оно должно было быть отображено) должны были быть нарисованы тонкими линиями.

В ранних версиях AutoCAD ширину линии, отличную от нулевой, можно было установить только для полилиний. При этом приходилось использовать специальные программы, которые автоматически устанавливали требуемую ширину полилиний, или "соображать", как это делать вручную (например, запоминать, что для этого масштаба "жирные" линии должны иметь ширину 50 единиц). Для изменения ширины линий приходилось физически редактировать свойства объектов. При этом уже затруднялось использование внешних ссылок и автоматического отслеживания изменений в смежных разделах проекта.

Широко использовался (и сейчас еще используется) прием, при котором все рисовалось тонкими линиями, и только при выводе на бумагу использовалась разная ширина пера, привязываемая к цвету примитивов. Но в этом случае нарушался принцип WYSIWYG (What you see is what you get) — "что вижу на экране, то и имею на бумаге", а в очень многих случаях требуется именно правильное изображение на экране.

В системе AutoCAD 2000 было внедрено новое замечательное свойство для всех примитивов — *вес линии* (lineweight). Этот термин не очень точно отражает суть

свойства, но более подходящие названия ширина или толщина уже были использованы. Фактически это ширина или "толщина" линий, которыми изображаются все объекты, включая тексты. Вес, как и другие свойства, может быть назначен как отдельному примитиву (в том числе со значением веса ByLayer (ПОСЛОЮ), так и слою. В отличие от физической ширины полилиний вес изображается на экране с заданным размером независимо от масштаба экранного изображения, а режим отображения весов может включаться и отключаться.

Помимо очевидных достоинств вес имеет некоторые недостатки. Для многих изображений требуется, чтобы часть их отображалась непременно "тонкими" линиями, а часть — тонкими или "толстыми". Раскладывать части таких изображений по разным слоям нерационально, а чаще вообще невозможно.

В отличие от рекомендаций по использованию цвета, применять для веса значение ByLayer (ПОСЛОЮ) нецелесообразно. Мы предусмотрим программное управление весом так, чтобы изображения, для которых линии всегда должны быть тонкими (например, осевые линии трубопроводов) рисовались с нулевым весом, а изображения, у которых может быть разная ширина линий (например, контуры трубопроводов), рисовались с заданной пользователем шириной. Изменять текущую ширину линий можно будет в любой программе, в которой это имеет смысл.

Как учитывать работу в пространствах листа и модели

Как показывает опыт общения с пользователями, все, кто разобрался с работой в пространствах листа и модели, осознают преимущества таких возможностей и в дополнительной агитации не нуждаются.

Большинство пользователей, работающих только с пространством модели, просто не разобрались с преимуществами этих давно уже не новых технологий, а иногда вообще о них не знают. Это, как правило, люди, работавшие в AutoCAD R14, в котором "лист" не сразу виден. Впрочем, бывают и объективные причины для работы только в пространстве модели, например требования заказчиков.

При разработке ruCAD мы не будем "учить жить", а "поможем материально", т. е. сделаем так, чтобы независимо от текущего рабочего пространства все масштабируемые символы рисовались именно такой величины, какая должна быть на бумаге при установленном масштабе чертежа. Для этого любая наша программа должна автоматически, с учетом текущего пространства правильно масштабировать все изображения. Если, например, пользователь будет рисовать рамку формата размером 297×420 мм при установленном масштабе чертежа 1:100, то при работе в пространстве листа размер рамки в единицах рисунка будет 297×420, а в пространстве модели — 29 700×42 000.

Для программной реализации такой возможности необходимо:

- при создании рисунка запросить у пользователя масштаб и единицы измерения;
- хранить эту информацию внутри рисунка;
- автоматически масштабировать символы во время работы.

В не столь отдаленные времена лучшим местом для хранения масштаба рисунка была системная переменная DIMSCALE, в соответствии с которой масштабировались все величины элементов размеров. В современных версиях системы AutoCAD это

удобное местечко перестало быть укромным и надежным. Связано это со способами нанесения размеров.

Нанесение размеров

Существует три способа построения размеров.

- *Нанесение размеров в пространстве модели для печати в пространстве модели*. Это стандартный способ, используемый при работе с одним видом рисунка. Для корректного масштабирования размеров при выводе на печать системной переменной DIMSCALE присваивается значение, обратное установленному масштабу печати. Только так работали в AutoCAD R10 и так работают те, кто не признает пространство листа до сих пор.
- *Нанесение размеров в пространстве модели для печати в пространстве листа*. Это рекомендуемый нами способ для строительных чертежей. В этом случае, возможно, для отключения вывода размеров одного видового экрана листа на других, потребуется создание отдельных слоев размеров для каждого видового экрана листа, замораживаемых на всех остальных видовых экранах. При значении переменной DIMSCALE, равной знаменателю масштаба чертежа, и при соответствующей установке масштабов видовых экранов в пространстве листа величина размерных элементов на бумаге будет соответствовать ГОСТ. Для автоматического масштабирования размеров при отображении в пространстве листа системной переменной DIMSCALE присваивается значение 0, а *размеры в пространстве модели надо наносить, находясь на вкладках листов*, а не модели. Переменная DIMSCALE становится непригодной для хранения масштаба.
- *Нанесение размеров в пространстве листа*. Это самый простой способ. Для построения размеров в пространстве листа выбираются объекты пространства модели или включается объектная привязка для позиционирования в точках объектов модели. По умолчанию устанавливается ассоциативная связь между размерами пространства листа и объектами модели. Системные переменные DIMLFAC и DIMSCALE устанавливаются равными 1. Недостаток этого способа в том, что размеры из пространства листа не будут передаваться другим пользователям при записи на диск только объектов из модели, а это часто требуется. Помимо того, этот механизм не действует при использовании примитивов ADT (Architectural Desktop).

Примечание

В системах AutoCAD 2002 и AutoCAD 2004 выполняется автоматический учет масштаба видового экрана и проставляются истинные размеры, даже когда значение системной переменной DIMLFAC равно 1. Однако не всегда (наверное, плавающая ошибка). Предполагаю, что это происходит из-за неправильной конвертации рисунка из DWG-формата AutoCAD 2002 в DWG-формат AutoCAD 2004. *(Николай Полещук)*

Иногда используется комбинированный способ — "общие" размеры (например, между координационными осями) создаются в пространстве модели, а "специальные", которые не нужно передавать другим пользователям — в пространстве листа.

В своей программной системе мы должны обеспечить пользователям возможность простановки размеров любыми удобными способами. Мы не будем вмешиваться в приемы простановки размеров (они превосходно продуманы в AutoCAD), но мак-

симально упростим эту работу. Пользователь не должен будет ничего знать о размерных переменных и стилях, все настройки будут выполняться автоматически в зависимости от принятого масштаба чертежа. На случай необдуманного изменения настроек "слишком умным" пользователем мы предусмотрим автоматическое восстановление облика размеров.

Замечание

Мы не будем предусматривать автоматическую простановку размеров (за очень редкими исключениями), как это делается в некоторых системах! В таких программах или автоматически создаются заранее определенные слои, или размеры наносятся на одном слое с основным изображением. И то, и другое плохо.

Как сохранять и восстанавливать настройки ruCAD

Для такой большой системы наверняка потребуется много параметров, которые надо сохранять и восстанавливать. Только относительно несложные программы не требуют инсталляции и ничего никуда не пишут. При проектировании способов хранения настроек своей программы надо решить три задачи:

- Что хранить.
- Где хранить (предварительно выяснив, где *можно* хранить, и где мы все-таки *будем* сохранять настройки).
- Как сохранять и читать. На постановочном этапе мы определимся с форматами хранения данных, а на этапе программирования должны будем предусмотреть соответствующие программные реализации.

Разберем эти вопросы "в порядке поступления".

Что хранить в настройках ruCAD

С тем, что хранить, мы будем разбираться в последующих главах. Однако, забегая вперед, скажем, что настроек будет достаточно много. К ним можно отнести следующие:

- общие параметры системы (имя корневого каталога системы, имена каталогов, которые могут располагаться в других местах, в том числе на сервере, имена приложений, используемых в качестве редакторов и просмотрщиков файлов различных типов и т. п.);
- параметры, общие для всех пользователей (меню, иллюстрации к программам, классификатор слоев, различные значения по умолчанию, параметры штриховок и типов линий и т. п.);
- личные параметры, запоминаемые для каждого пользователя;
- параметры отдельных программ;
- параметры, специфичные для каждого проекта. В том числе дополнительные данные для папок и документов (комментарии, реквизиты и пр.);

- специфичные настройки для рисунков (стили текстов, размеров и т. п.);
- временные файлы, которые обычно генерируются вызывающей программой для передачи большого количества параметров вызываемой программе и для получения результатов работы.

О специальных папках Windows

Наши программы будут функционировать в среде Microsoft Windows и должны подчиняться "правилам игры", устанавливаемым операционной системой. Относится это и к организации каталогов программы. В Windows используется особая система специальных папок. Папка, в которой установлена Windows, уже является специальной. Обычно это C:\Windows, но может быть и другое имя (C:\WinNt, C:\WinXP и даже C:\Форточки). Из множества специальных папок нас будут интересовать только некоторые (табл. 3.3).

Таблица 3.3. *Специальные папки Windows*

Псевдоним	Назначение
WindowsFolder	Папка, в которую установлена Windows
WindowsSystemFolder	Системная папка Windows
WindowsTempFolder	Папка для временных файлов Windows
ProgramsFolder	Папка для программ
ProgramFilesFolder	Папка для установки программ
PersonalFolder	Папка для личных данных
CommonFilesFolder	Папка для общих данных программ
CommonProgramsFolder	Папка для общих программ
CommonAppdataFolder	Папка для общих данных приложений
AppdataFolder	Папка для данных приложений
CurrentFolder	Текущая папка

Некоторые программы (в том числе AutoCAD) используют собственные специальные папки. Мы тоже разработаем для ruCAD систему специальных папок.

Все специальные папки (включая папки AutoCAD и ruCAD) мы будем именовать псевдонимами, обрамленными символами %, например, %PersonalFolder%. Псевдонимы в последующем мы будем вводить без дополнительных пояснений, стараясь выбирать понятные псевдонимы, такие как AcadSupportFolder, ruCADAllUsersFolder. Кроме псевдонимов папок, мы будем использовать псевдонимы пользователей, например CurrentUserName.

К специальным папкам никогда не следует обращаться по их физическим именам. Если даже разработчик "всю жизнь" встречал папку C:\Мои документы, это не означает, что у всех пользователей и всегда будет так. И даже если вы знаете, что путь к PersonalFolder записан в раздел реестра

```
HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\
Shell Folders\Personal,
```

читать эти данные напрямую из реестра не следует, т. к. в другой конфигурации Windows эти данные могут оказаться в ином месте. Работать с именами специальных папок следует только через функции Windows API, возможно, заключая их для упрощения работы в свои функции-оболочки.

Где можно хранить настройки

Большинство программистов стремится сохранять параметры в файлах различных форматов в каталоге программы. Так бы и мы делали, если бы разрабатывали ruCAD только для использования в Windows 98/ME. Так же делала и система AutoCAD 2002. Когда-то это было верным решением, но сейчас, с переходом на новые многопользовательские операционные системы, ситуация усложнилась. Системы AutoCAD 2004 и AutoCAD 2005 устанавливаются гораздо изощренней. В традиционном для них каталоге Support уже нет прежнего обилия файлов, зато имеется файл Where are my Support Files.lnk, при щелчке по которому выводится справка системы AutoCAD с разделом, поясняющим, где теперь проживают файлы поддержки. Изучить расположение можно на вкладке **Files** (Файлы) диалогового окна **Options** (Настройка), вызвать которое можно, выбрав пункт **Options** (Настройка) меню **Tools** (Сервис). Многие файлы теперь находятся в каталогах

```
%AcadUserAppData%\Support\
%AcadUserAppData%\Recent\
%AcadUserAppData%\Plot Styles\
%AcadUserAppData%\Plotters\
%AcadUserAppData%\Data Links\
```

где `%AcadUserAppData%` — c:\Documents and Setings\%CurrentUserName% \Application Data\Autodesk\AutoCAD 2004\R16.0\enu или рядом с этим каталогом.

Другие файлы инсталлируются в каталоги

```
%AcadUserLocalAppData%
%AcadUserLocalAppData%\Template\
%AcadUserLocalAppData%\WebServices\
%AcadUserLocalAppData%\Textures\
```

где `%AcadUserLocalAppData%` — c:\Documents and Settings\%CurrentUserName% \Local Settings\Application Data\Autodesk\AutoCAD 2004\R16.0\enu.

Объясняется это тем, что в операционных системах Windows NT/2000/XP поддерживается определенная политика безопасности. Подробно описывать ее в этой книге мы не будем (тем более что решения Microsoft в этом направлении постоянно совершенствуются), но беглое знакомство необходимо.

Каждый пользователь имеет учетную запись, которая определяется именем пользователя и паролем, который он вводит при регистрации. Операционная система контролирует, отслеживает и ограничивает доступ к системным ресурсам, основываясь на разрешениях и правах, связанных с группами, в которые входит учетная запись пользователя или непосредственно с самой учетной записью. Наибольшие права на управление локальным компьютером имеют пользователи, входящие в локальную группу *Администраторы* (Administrators).

Пользователи разделяются на несколько групп: *Администраторы (Administrators)*, *Опытные пользователи (Power Users)*, *Пользователи (Users)*, *Гости (Guests)* и некоторые другие. Каждая группа имеет определенные привилегии (*Администраторы* могут их изменять). Системные администраторы дополнительно ограничивают привилегии обычных пользователей. В результате не каждый пользователь сможет писать данные, куда ему захочется.

Программисты часто не учитывают эти факты (на своем-то компьютере они всегда *Администраторы*) и потом удивляются, почему программы не работают на других рабочих местах. Для работы системы AutoCAD 2002 требуется, чтобы пользователь имел привилегии в операционной системе не ниже *Опытного пользователя (Power User)*, в том числе и потому, что его пользователь пишет[1] свои настройки не туда, "куда положено". Это очень не нравится системным администраторам, и они всегда ищут пути для ограничения прав пользователей AutoCAD[2].

В "нормальной" системе (Windows NT/2000/XP) программы запускаются от имени и с правами конкретного пользователя. Если этот пользователь не является *Администратором*, он имеет право изменять содержимое следующих ресурсов:

- часть реестра `HKEY_CURRENT_USER\` — домашний каталог;
- временный каталог (который, как правило, находится внутри домашнего);
- каталоги, специально выделенные для этого администратором.

Любые другие "подходящие" места могут оказаться доступными только для чтения, либо вообще недоступны. Например, совсем не обязательно, что будет доступен каталог Program Files или Windows. Кроме того, возможно, что инсталляцию систем AutoCAD и ruCAD будет выполнять *Администратор*, а работать с ними будет *Обычный пользователь*.

В соответствии с рекомендациями Microsoft в Windows имеется три места, предназначенных для хранения настроек, которыми и следует пользоваться:

- системный реестр;
- общий каталог для пользователей;
- домашний каталог пользователя (или один из его подкаталогов).

Системный реестр

Это то самое место, куда рекомендует записывать данные Microsoft, и куда очень не любят писать что-то большинство программистов. Причин этого прохладного отношения много. Наиболее характерные из них следующие:

- с одной стороны, хранение параметров в реестре — один из самых надежных способов с быстрым стандартным доступом, встроенной системой разграничения доступов и возможностью реализации моментального восстановления (при минимальных предварительных усилиях); с другой стороны, данные, находящиеся в реестре, могут быть утеряны в результате сбоев в Windows;

---

[1] Выражение "пользователь пишет" означает также и "пишет программа, запущенная от имени пользователя".

[2] AutoCAD 2004 и 2005, сертифицированные для работы с Windows XP, уже не требуют для работы прав Power User.

- с одной стороны, иерархическая структура реестра облегчает поиск нужного раздела; с другой стороны, это увеличивает риск удаления данных из-за неосторожных действий пользователей при непродуманной системе хранения данных;
- реестр постоянно "разбухает" (только Microsoft Office увеличивает его размер сразу на 3 Мбайта), в нем, даже при "штатном" удалении программ, остается огромное количество "мусора", а его размеры ограничены в настройках Windows. Очень большие файлы реестра снижают скорость загрузки операционной системы.

С точки зрения хранения настроек программ, системный реестр разделен на две части:

- `HKEY_CURRENT_USER` — для хранения настроек, специфичных для пользователя;
- `HKEY_LOCAL_MACHINE` — для хранения настроек, специфичных для всего компьютера и, соответственно, для всех пользователей, работающих на компьютере. Для этой части надо рассчитывать на то, что данные ее ветвей будут доступны пользователю только для чтения.

Настройки рекомендуется хранить в ветвях:

```
HKEY_CURRENT_USER\Software\%Company Name%\%ApplicationName%\%Version%
```

и

```
HKEY_LOCAL_MACHINE\Software\%Company Name%\%ApplicationName%\%Version%.
```

Домашний каталог пользователя

Для хранения настроек, слишком больших для того, чтобы их размещать в реестре, существуют специальные папки внутри домашнего каталога пользователя. Эти каталоги имеют имена Application Data и Local Settings. Полный путь к этим каталогам следует получать с помощью функций Windows API.

Общий каталог пользователей

Обычно это каталог Documents and Settings\All Users. Внутри него имеются такие же подкаталоги для хранения настроек и данных программ, но относящихся ко всем пользователям. Полный путь к ним можно также получить с помощью функций Windows API.

В каталогах пользователей данные можно хранить в любом формате.

Где мы будем хранить настройки

Ранее мы рассмотрели, где рекомендует хранить данные фирма Microsoft.

Места для размещения настроек нужно выбирать и с учетом возможностей доступа к ним собственных программ. Например, для работы с реестром из LISP-программ имеются стандартные функции, для работы же с INI-файлами потребуется разработка собственных функций. Проще всего было бы воспользоваться рекомендациями Microsoft, не "изобретать велосипед", и все записывать в реестр, но "мы пойдем другим путем".

Совершенно очевидно, что размещения данных в системном реестре избежать не удастся. Как минимум, там будут находиться регистрационные данные COM-серверов, которые мы будем создавать. Кроме того, в реестр, независимо от нашего

желания, будут записывать данные некоторые компоненты и формы, фиксирующие там свое размещение на экране. В реестре целесообразно разместить основные данные для системы ruCAD в целом. Для этого мы создадим ветвь реестра

```
HKEY_CURRENT_USER\SOFTWARE\ruCAD group\ruCAD
```

в которой будем создавать дополнительные ключи для хранения имен каталогов системы.

Решение по размещению остальных настроек не может быть однозначным. Мы, например, можем установить системное требование о том, что доступ ко всем каталогам ruCAD должен быть полным для всех пользователей. Это вполне обоснованное решение, т. к. и сам AutoCAD, и приложения к нему являются основными программами, для которых и предназначен компьютер (в отличие, например, от множества "факультативных" утилит). В этом случае значительно упрощается сопровождение и обновление системы. Очень трудно эксплуатировать систему, у которой одни и те же данные на одном компьютере хранятся в каталоге C:\Windows \Application Data, а на другом — сразу в нескольких каталогах.

Однако пуристы[1] идеологии Microsoft, которые наверняка могут оказаться среди системных администраторов, вполне могут потребовать этого. Очевидно, следует пойти на компромисс. Мы, на этапе разработки, будем размещать все файлы в подкаталогах одной корневой папки, но группировать их так, чтобы была возможность без переделки программ "разбросать" их в требуемые места. Для облегчения таких манипуляций мы будем именовать собственные каталоги таким образом, чтобы было ясно, куда этот каталог *можно* переместить. Сценарий установки системы *(см. главу 36)* мы будем разрабатывать с учетом требований системы безопасности.

Нам также придется ориентироваться на идеологию системы AutoCAD 2004, которая сертифицирована для работы с Windows XP, не требует для работы прав Power User, и соответствует требованиям системы безопасности в части размещения файлов.

Примечание

Если проанализировать, что делает система AutoCAD 2004 при инсталляции, то можно прийти в ужас. LOG-файл инсталлятора имеет размер более 4 Мбайт, компоненты системы устанавливаются более чем в сотню папок, включая папки \Common Files\Microsoft Shared\Office10\ и подобные, в реестре регистрируется несколько тысяч компонентов.

Для системы ruCAD мы определим следующие каталоги:

- %ruCADRootDir%\LocalSettings\All Users\ruCAD — для параметров, общих для всех пользователей;
- %ruCADRootDir%\LocalSettings\%CurrentUser%\ruCAD — для личных параметров,
- %ruCADRootDir%\LocalSettings\Application Data\ruCAD — для параметров отдельных программ.

При необходимости эти каталоги можно будет перенести в специальные папки Windows.

Дополнительные данные для папок и документов, естественно, будут размещаться в тех же каталогах, где и документы. Даже сисадмины с их увлеченностью безопас-

[1] Люди, исповедующие принципы в "чистом" виде (pure — чистый).

ностью должны понимать, что программы для того и существуют, чтобы создавать документы[1].

Мнение профессионального сисадмина

> Не следует забывать, что несмотря на то, что профессиональный сисадмин, с точки зрения абсолютного большинства пользователей, является злобным параноиком, только мешающим работе, на самом деле, без этой "паранойи" ни одна сколь-нибудь серьезная сетевая система существовать не может. Очень показательно то, что чем больше пользователь озабочен выполнением своих основных обязанностей, тем меньше он склонен к истерикам по поводу "попранной свободы". Типичное мнение сисадминов о разработчиках, не учитывающих вопросы разграничения прав пользователей, не может быть приведено здесь по цензурным соображениям. *(П. Лоскутов)*

Мы специально привели дословный комментарий подлинно высокопрофессионального системного администратора Петра Лоскутова для того, чтобы разработчики представляли серьезность вопроса. Петр не только грамотный системный администратор, но и великолепный программист для AutoCAD, т. е. знает дело со всех точек зрения и не заинтересован в надуманных ограничениях.

О временных файлах

Естественным местом для хранения временных файлов является специальный системный каталог. Имя его возвращается системной функцией `GetTempPath`. Но при ее использовании могут быть опасные коллизии. К очень серьезным последствиям может привести программное удаление из него всех файлов. В Windows 95/98/ME временный каталог ищется сначала в переменной окружения TMP, затем в переменной окружения TEMP, но если эти переменные не определены, то временным каталогом считается текущий. В Windows NT/2000/XP еще опасней. Если не определены переменные TMP и TEMP, то каталогом для временных файлов считается каталог Windows. Практика показывает, что иногда "темповые" каталоги забывают создавать или (конечно из лучших побуждений) удаляют. Для своей системы мы создадим специальный временный каталог. Это будет очень удобно на этапе разработки системы, т. к. все временные файлы будут находиться в доступном месте, а в отладочных режимах мы их вообще не будем удалять для того, чтобы наслаждаться интимными подробностями жизнедеятельности наших программ.

Как сохранять и читать настройки

Работать с системным реестром мы будем не иначе, как с использованием функций Windows API, которые в неявном для программиста виде вызываются даже тогда, когда мы применяем встроенные функции любой среды программирования. Здесь и выбирать не из чего.

INI-файлы

В пользовательских каталогах данные можно хранить в любом виде, но чаще всего используются INI-файлы. Настройки хранятся в текстовом файле так, как это показано в листинге 3.2.

---

[1] Системный администратор добавил бы: "Но только в отведенных для этого местах".

Листинг 3.2. Фрагмент INI-файла

```
[Имя секции 1]
; Комментарий
Имя переменной 1=Значение переменной 1
Имя переменной X=Значение переменной X
[Имя секции X]
; Комментарий
Имя переменной 1=Значение переменной 1
Имя переменной X=Значение переменной X
```

Преимущества INI-файлов по сравнению с реестром в защищенности от сбоев операционной системы, в легкости просмотра и редактирования с помощью обычного текстового редактора, а также мобильности. При некоторой изворотливости программиста в INI-файле можно хранить данные любого типа, даже растровые изображения. Разумеется, любые данные преобразуются при записи в текстовый вид. В языке Visual LISP, на котором мы будем писать большинство программ, нет встроенных функций для работы с INI-файлами, но их легко написать самим.

Именно INI-файлы мы и будем чаще всего использовать.

Использование XML

Многим программистам хочется иметь иерархические хранилища данных, такое же, как системный реестр, но размещаемое в отдельных файлах. Увы, Microsoft не дает такой возможности. Однако имеется несколько альтернативных технологий.

Одной из них является использование расширяемого языка разметки Extensible Markup Language, или XML. Возможности его очень широки — от использования для размещения информации в World Wide Web до хранения графических данных. Система AutoCAD 2002 уже могла сохранять и читать блоки (а значит, и сами рисунки) в формате DesignXML, но из AutoCAD 2004 эта возможность исключена. DesignXML в системе AutoCAD 2004 используется во внутренних целях. Скорее всего, XML сможет стать универсальным языком для хранения любых документов, но Autodesk, несмотря на недавние лозунги о стратегической поддержке XML для представления данных, "сделала ручкой" этой технологии (именно так, в переводе на русский язык, выражаются в Интернете об этих шараханьях)[1]. Изучать XML в этой книге мы, конечно, не будем, но возможностями его будем пользоваться широко.

В формате XML мы будем хранить самые сложные настройки — иерархические иллюстрированные меню и классификатор слоев. Этим технологиям посвящена значительная часть книги. В настоящее время для хранения простых настроек мы XML применять не будем, потому что пока нет стандартных средств для работы с XML, как с системным реестром. Многие разработчики, в том числе и мы, уже создали собственные инструменты для хранения настроек в XML.

[1] Создается впечатление, что в Autodesk просто испугались слишком хороших возможностей DesignXML для изучения графической базы данных рисунка.

Политика работы с пользователями в ruCAD

В нашей системе явно потребуется поддержка определенной (хотя и не такой сложной) системы работы с пользователями ruCAD. Теоретически на каждом *персональном компьютере* должен работать только один человек. Практически, особенно в России, такое бывает не всегда. Часто на одном компьютере попеременно работают разные люди. Причем, это бывает не только из-за бедности проектной организации, но и при нормальной работе в несколько смен. Но даже если на компьютере работает только один человек, определенная политика безопасности все равно требуется, т. к. люди имеют разную квалификацию (от "чайника" до "профи"). Желательно установить некоторые ограничения, чтобы какая-нибудь "обезьяна с гранатой" не могла, даже случайно, испортить важные файлы. Кроме того, при организации электронного архива потребуются (помимо основной системы безопасности Windows) некоторые специфические ограничения и привилегии.

Помимо вопросов безопасности имеют значение и дополнительные удобства для пользователей. Например, очень хорошо, когда каждый конкретный человек имеет собственный рабочий каталог, в котором ему предлагается начинать выбор файлов, когда он имеет собственное меню, соответствующее его специализации, собственный словарь и прочие "вкусности". Даже сообщение об ошибке, с указанием в заголовке имени конкретного человека, воспринимается лучше. "Мелочь, а приятно".

Для реализации такой политики потребуются некоторые дополнительные усилия.

- Ведение реестра пользователей с запоминанием для каждого пользователя имени, пароля, рабочего каталога, специальности и других параметров.
- Регистрация пользователя при запуске ruCAD.
- Установка разрешений для пользователя на различные действия и проверка их во время работы.

Как организовать систему папок и документов

Настало время продумать и создать файловую систему ruCAD. Прежде всего, следует отказаться от идеи (если она у вас возникла) положить все куда-нибудь в каталоги самого AutoCAD, например в Support. Мы будем держаться от самого AutoCAD подальше и не будем размещать в его каталогах ни одного файла. Это сделает нашу систему полностью независимой от какой-либо версии AutoCAD. При формировании структуры каталогов мы будем учитывать, что нам потребуется место для исходных текстов программ, которые не будут передаваться конечным пользователям, для файлов, которые могли бы размещаться на сервере, для хранения различных настроек (о чем мы уже рассуждали в этой главе). В целом мы будем стараться "разложить все по полочкам", размещая в отдельных каталогах файлы различного логического назначения.

Корневой каталог системы

В первую очередь определим, где будет находиться корневой каталог. Рекомендуемым стандартным местом для размещения всех программ является каталог Program Files. Именно туда и следует предлагать устанавливать ruCAD при инсталляции на

компьютеры пользователя. Плохим тоном является предложение устанавливать программы в корневой каталог диска C:. Но на этапе разработки лучше разместить ruCAD в корневой каталог любого диска, просто чтобы не лазить тысячи раз (а это придется делать) в глубоко спрятанную папку. В книге мы будем считать, что разместили ruCAD на диске C:.

Давайте присвоим имя нашему каталогу c:\.ru (обратите внимание на точку перед именем). Такое необычное имя, содержащее расширение без основной части, вполне допустимо. Каталог с таким именем будет всегда располагаться (при сортировке по именам) в самом верху списка папок, что значительно ускорит нашу работу.

Саму систему мы разместим в каталоге c:\.ru\CAD. Это и будет наш "ruCADRootDir", который мы запишем в системный реестр. Обратите внимание, наш %ruCADRootDir% к тому же и соответствует нашему рабочему логотипу!

Рядом с подкаталогом CAD можно размещать и другие вспомогательные каталоги. Например, у нас рядом размещены каталоги Book (в котором мы складируем файлы с текстом этой книги), Install (в котором накапливаются дополнительные файлы, потребующиеся впоследствии для создания инсталляции) и некоторые другие.

Соглашение об именах файлов

Давайте договоримся об именовании файлов и каталогов. Мы будем применять исключительно имена с латинскими символами, но не будем придерживаться устаревшей системы коротких имен в формате 8.3. Русские имена мы применять не будем потому, что рано или поздно очередная версия каких-нибудь утилит посчитает их "неправильными" и переведет в нечитаемый вид.

Для файлов, которые будут создавать пользователи, мы будем рекомендовать (но не категорически) придерживаться такой же системы. Впоследствии мы создадим специальный файловый навигатор, который позволит поддерживать систему длинных русских комментариев к файлам и папкам.

Совет

Пока мы не сделали свой навигатор, рекомендуем использовать в качестве файлового менеджера программу Total Commander (бывший Windows Commander) версии не ниже 5.5. Эта программа позволяет вести комментарии к файлам в известном формате descript.ion. Для просмотра комментариев нужно включить в меню **Show** (Вид) опцию **Comments** (Комментарии) или нажать горячие клавиши <Ctrl>+<Shift>+<F2>. Комментарии создаются и редактируются для выделенного файла или каталога при нажатии комбинации клавиш <Ctrl>+<Z> или при выборе пункта **Edit Comment** (Редактировать комментарий) в меню **File** (Файл). В результате в каталоге автоматически создается файл descript.ion, в котором хранятся примечания к файлам и каталогам. При переименовании файлов автоматически синхронизируется и descript.ion.

Total Commander пригодится вам и для других операций (помимо копирования, переименования и прочего), например, для синхронизации каталогов, периодического архивирования измененных файлов и т. п.

Каталоги AutoCAD

Нам понадобятся три каталога для файлов нашей системы, используемых AutoCAD (меню, описания штриховок и типов линий, пиктограммы для панелей инструмен-

тов, файлы шрифтов и т. п.). Два каталога будут использоваться для файлов версий системы AutoCAD 2002 и AutoCAD 2004, а третий — для файлов, не зависящих от версии. Эти каталоги обязательно должны включаться в Support Search Path для нашего профиля, причем на первое место. Такой прием позволит нам переопределять и стандартные файлы AutoCAD, не затрагивая оригиналов. В то же время мы не будем складывать в эти каталоги файлы LISP-программ и диалоговых окон (обычно размещаемые на путях поиска AutoCAD). Это мы сделаем для того, чтобы "голый" AutoCAD, запущенный "просто так" с нашим профилем, не мог даже случайно загружать наши программы (все равно они без предварительной подготовки не смогут работать).

Итак, создаем следующие каталоги:

- %ruCADRootDir%\Local Settings\Application Data\ruCAD\AutoCAD\ — для общих компонентов (псевдоним %ruCadLocalAppDataAcad%);
- %ruCadLocalAppDataAcad%\15 — для компонентов версии 2002;
- %ruCadLocalAppDataAcad%\16 — для компонентов версии 2004.

Каталог Bin

В каталоге %ruCADRootDir%\Bin мы будем размещать собственные исполняемые файлы, DLL, библиотеки типов и ресурсы, которые должны проживать рядом с исполняемыми файлами.

Каталог Temp

В каталоге %ruCADRootDir%\Temp будут находиться временные файлы, о которых мы уже говорили в этой главе.

Каталог Source

В каталоге %ruCADRootDir%\Source и его подкаталогах (LISP, Delphi, ObjectARX, Help и т. п.) будут храниться исходные тексты программ, справочной системы и другие материалы, предназначенные для использования только разработчиками. Этот каталог не будет включаться в список включаемых в инсталляционный комплект.

Каталог Samples

В каталоге %ruCADRootDir%\Samples и его подкаталогах мы будем размещать примеры DWG-файлов.

Каталог All Users

В каталоге %ruCADRootDir%\All Users и его подкаталогах мы будем размещать данные, общие для всех пользователей. При необходимости этот каталог может быть перемещен со всеми вложениями на другое место, в том числе и на сервер. Внутри каталога All Users мы создадим ряд следующих подкаталогов:

- App — для откомпилированных LISP-программ;
- Block — для блоков, сохраняемых в отдельных файлах;

- Block-Lib — для файлов блоков-библиотек, внутри которых определено много маленьких блоков по определенной тематике;
- Block-Lw — для файлов блоков с изменяемой шириной линий;
- Dcl — для файлов описания диалоговых окон;
- Ini — для INI-файлов различного назначения;
- Help — для справочных файлов различных форматов;
- Table — для файлов шапок таблиц и их описаний;
- Template — для шаблонов-заготовок различных файлов;
- Txt — для текстовых файлов типовых примечаний, указаний, технических требований, вставляемых в рисунки;
- Xml — в этом каталоге и его подкаталогах будут находиться XML-файлы и требуемые для них ресурсы:
  - Class — классификатор слоев;
  - Dic — словари, используемые при редактировании XML;
  - Menu — меню, данные и опции программ в формате XML;
  - Images — иллюстрации к меню в формате XML.

Внутри этих каталогов, при необходимости, могут создаваться тематические подкаталоги.

Соглашение

Впредь этот каталог в книге мы будем называть %ruCADAllUsersDir%. Путь к нему будет записан в реестр.

Каталог Current User

В каталоге %ruCADRootDir%\Current User и его подкаталогах будут размещаться файлы со специфичными для конкретного пользователя данными и настройками.

Каталог Application Data

В каталоге %ruCADRootDir%\Application Data и его подкаталогах будут размещаться файлы со специфичными настройками и данными отдельных программ.

Регистрация каталогов

Для регистрации в системном реестре расположения папок системы в обычном текстовом редакторе создаем файл ruCAD.reg (листинг 3.3) и "выполняем" его.

Листинг 3.3. Файл ruCAD.reg

```
REGEDIT4

[HKEY_CURRENT_USER\SOFTWARE\ruCAD group]

[HKEY_CURRENT_USER\SOFTWARE\ruCAD group\ruCAD]
"RootDir"="C:\\.ru\\CAD"
```

```
"AppDataDir"="C:\\.ru\\CAD\\Application Data"
"LocalSettingsDir"="C:\\.ru\\CAD\\Local Settings"
"LocalAppDataDir"="C:\\.ru\\CAD\\Local Settings\\Application Data\\ruCAD"
"LocalAcadAllVersionDir"="C:\\.ru\\CAD\\Local Settings\\Application Data\\ruCAD
\\AutoCAD"
"AllUsersDir"="C:\\.ru\\CAD\\All Users"
"CurrentUserDir"="C:\\.ru\\CAD\\Local Settings\\Current User"
"LspSourceDir"="C:\\.ru\\CAD\\source\\LISP"
```

Замечание

Такое расположение каталогов мы предусматриваем на этапе разработки. На компьютерах пользователей программа установки будет размещать компоненты системы (и делать соответствующие записи в реестр) "куда положено", с учетом требований системы безопасности.

Какие программы потребуются для реализации базовых принципов

Для реализации изложенных базовых принципов потребуется мощная программная поддержка. Реализация программной поддержки должна быть предусмотрена в библиотечных функциях и специальных приложениях. Конкретные программы для рисования объектов и управления системой AutoCAD должны использовать библиотечные функции. Исходя из разработанной концепции, мы планируем разработку следующих программ и библиотек.

Программа-стартер

Программа-стартер должна осуществлять идентификацию пользователя, установку его предпочтений, привилегий и разрешений. В стартере должны выполняться основные настройки системы и реестра пользователей, а также должна выбираться рабочая система AutoCAD (вероятно, на компьютере их имеется несколько). Из стартера должна запускаться рабочая система AutoCAD (возможно, с предварительным выбором документов для загрузки). Стартер должен перед запуском AutoCAD генерировать файл для автоматической загрузки требуемых библиотек. Стартер может выступать в качестве внешнего COM-сервера, к объектам и методам которого можно обращаться из LISP-программ.

ARX-библиотека

Эта библиотека будет разрабатываться в среде Visual C++ с использованием библиотеки ObjectARX. Функции этой библиотеки будут доступны в Visual LISP. Вообще-то без такой библиотеки можно обойтись, а требуемые функции разработать с использованием других сред программирования (Delphi, VBA), но, т. к. создание ARX-приложений является основным способом расширения возможностей Visual LISP, рекомендованным Autodesk, мы разработаем такую библиотеку.

COM-серверы

Некоторые функции мы реализуем с использованием COM-технологий. Они широко используются в Visual LISP при работе с объектной моделью системы AutoCAD. Из Visual LISP можно обращаться к любому COM-серверу, не только к AutoCAD. В литературе часто приводятся примеры работы с готовыми COM-серверами (обычно Microsoft Word или Excel), мы же приведем примеры разработки COM-серверов в нетрадиционной для AutoCAD среде программирования Borland Delphi. Это будут не просто учебные примеры, а реально работающие приложения, которые мы будем широко использовать в ruCAD. С использованием COM-технологий мы реализуем следующие самые сложные функции системы:

- диалоги выбора файлов и папок с использованием предварительного просмотра рисунков, возможностью редактирования примечаний к файлам и папкам, просмотром общих и специальных свойств файлов;
- диалоги выбора слоев, программ, их опций и других данных из иллюстрированных иерархических меню в формате XML;
- различные диалоги, реализация которых средствами DCL невозможна.

LISP-библиотеки

В библиотеках, написанных на языке Visual LISP, будут сосредоточены все функции системы. Отличие библиотек от обычных программ в том, что они при загрузке ничего не делают. Библиотечные функции вызываются из основных программ. Исходя из разработанной концепции ruCAD, нам потребуется разработка следующих основных семейств функций для:

- определения расположения компонентов системы;
- поддержки системы пропорциональности;
- инициализации системы, настройки параметров рисунка и переключения режимов работы;
- выбора программ и их опций из XML-деревьев;
- работы с меню;
- загрузки программ;
- обработки ошибок;
- вывода диалоговых окон, в том числе с помощью COM-серверов;
- проверки прав пользователей во время работы;
- работы с классификатором слоев;
- работы с объектной моделью AutoCAD;
- управления свойствами объектов;
- преобразования типов данных;
- геометрических построений;
- эффективного использования блоков;

- унификации ввода данных;
- обработки строк, текстов и списков;
- рисования типовых объектов (линий, контуров, трасс, таблиц);
- типовых вычислений (площади, длины, объемы).

Все эти функции физически будут размещаться в нескольких файлах, собираемых во время компоновки приложения в одну главную библиотеку, загружаемую в каждый рисунок.

Кроме того, потребуется разработка нескольких специальных библиотек, загружаемых только при вызове определенных программ, например, библиотек для рисования окон, лестниц, вентиляторов и т. п.

ГЛАВА 4

Миграция из ранних версий AutoCAD

При разработке вам могут пригодиться программы и рисунки, выполненные в устаревших версиях системы AutoCAD. Это могут быть и собственные программы, и "закачанные" из Интернета, и коллекции блоков, и чертежи, извлеченные из старых архивов. Хотя фирма Autodesk декларирует возможность использования "древних" данных, верить этому не стоит. Проблемы будут, и проблемы серьезные. Даже если вы делаете все заново, пользователи непременно столкнутся с проблемами использования рисунков, выполненных в ранних версиях. Мы должны дать им инструменты для таких операций.

Переход от одной Windows-версии AutoCAD к другой (обычно от R14 к 2000/2000i/2002), на первый взгляд, не представляет особых проблем. На самом деле это не так просто. Механический перенос LISP-программ возможен только для простых случаев, а вот библиотеки функций, содержащиеся в ARX-файлах, уже будут принципиально несовместимыми.

Как осуществлять переход под Windows

Еще до сих пор некоторое количество пользователей работает в операционной системе MS DOS с использованием AutoCAD R10 и R12. Иногда это происходит по экономическим причинам — недостаток средств для обновления компьютеров и программного обеспечения, иногда просто из-за того, что все требуемые задачи прекрасно решаются имеющимися программными средствами, а аналогов, работающих под Windows, просто нет. Возможно, что перевод старых наработок в современную операционную систему и будет вашей задачей.

Такая задача потребует и своей постановки, и своего плана. Прежде всего следует выяснить, стоит ли заниматься "дословным" переводом. Если пользователи "засиделись", скажем, в AutoCAD R10, то вполне возможно, что их любимые задачи уже реализованы в современных версиях системы AutoCAD или в приложениях к нему. Все-таки за последние годы появилось много разработок, в том числе для российских пользователей.

Если перевод требуется, то его нужно разделить на подзадачи.

Миграция приложений, написанных на AutoLISP

Для этого, как минимум, потребуется перекодировка текстов программ из кодовой таблицы dos866 в win1251. Когда-то это было проблемой, но теперь имеется несметное количество программ-перекодировщиков, да и многие текстовые редакторы позволяют, открыв файл в одной кодировке, сохранить его в другой. Это самая легкая часть работы, а более сложные вопросы мы рассмотрим далее.

Миграция внешних приложений, работающих в DOS

Многие системы, работавшие под AutoCAD в DOS, включали в себя внешние DOS-программы, использовавшиеся для работы с данными, составления спецификаций и других "благородных" целей. Наверняка все их придется выбросить навсегда, а взамен написать современные аналоги. Конечно, можно попытаться запускать DOS-программы из современной системы AutoCAD, иногда это будет получаться, но это следует рассматривать только как временную "расшивку узких мест".

Перевод внешних программ под Windows — очень большая работа. Программировать в Windows с использованием современных инструментальных средств разработки гораздо проще. Приспособить старые тексты и алгоритмы можно с большим трудом, даже если в современных системах поддерживаются аналогичные языковые конструкции. Не пытайтесь перейти на Windows-аналоги устаревших и умирающих систем, таких как Clipper, используйте самые перспективные среды разработки. Наверняка придется изменить и идеологию работы прикладных приложений. Например, если в прежней системе из AutoCAD запускался EXE-файл для выполнения расчетов, а после завершения работы программы вновь активизировалась система AutoCAD, то теперь такой расчетный модуль может быть реализован в виде COM-сервера в DLL.

Миграция библиотек блоков

Безусловно, старых блоков накопилось много. Если это простые изображения, не имеющие никаких атрибутов и других именованных объектов на русском языке, то вопрос решается просто — загрузить в AutoCAD и сохранить в современном формате. Если же внутри имеются русские символы, то конвертирование придется выполнять так же, как и конвертирование основных рисунков.

Шрифты

Вам придется обзавестись комплектом шрифтов в WIN-кодировке. Проблемы с этим давно нет, но проблема может быть с именами шрифтов. Имена стандартных и многих дополнительных файлов шрифтов для DOS и Windows одинаковые, а внутри рисунков имеются ссылки на файл шрифта. Выбрасывать DOS-шрифты не следует, они вам еще пригодятся. Их просто нужно переименовать (например, txt.shx в dos_txt.shx) и положить в каталог шрифтов "новой" системы AutoCAD.

Штриховки и типы линий

Если имеются файлы с нестандартными штриховками и типами линий, их также следует перекодировать, как и другие текстовые файлы. Если внутри этих файлов

нет никаких русских символов (хотя бы в комментариях), их можно использовать сразу.

Файл acad.pgp

В этом файле описываются команды, запускающие внешние программы. В DOS-версиях этот файл имел очень большое значение, т. к. только через него можно было запускать "из-под" системы AutoCAD, во время ее работы, самостоятельные программы. Иногда в нем описывались очень хитроумные комбинации. В многозадачной среде Windows этот файл потерял свое прежнее значение. Теперь он используется более для описания коротких псевдонимов команд. Тащить его за собой из старых версий нет смысла, разве что скопировать из него привычные вам псевдонимы команд. Разумеется, для этого потребуется перекодировка файла.

Меню

Меню старой системы AutoCAD, как минимум, потребует перекодировки. Фактически же вам его потребуется полностью переписать с учетом новых возможностей и, вероятно, имен команд и их опций.

Базы данных

Если старые программы использовали базы данных, как "настоящие", в форматах dBase или Paradox, так и в различных текстовых форматах, то наверняка потребуется их изменение. Текстовые файлы можно просто перекодировать, хотя стоит подумать и о реорганизации хранения данных. "Настоящие" базы данных, если они не используются какими-то другими приложениями, лучше перевести в форматы современных СУБД. Однозначных рекомендаций дать нельзя, но очень удобны базы данных Microsoft Access. Вообще в Windows имеется большой выбор СУБД на все вкусы и потребности. В отличие от DOS, где часто использовался прямой доступ к базам данных, в Windows имеется достаточный выбор "движков" (DataBase Engines) для работы с данными любых форматов. Можно работать и с базами данных в DOS-кодировке.

Как выполнять перекодировку рисунков

Массовая, при переходе на работу под Windows, или периодическая — при использовании старых рисунков — перекодировка будет требоваться еще долго. Операции по перекодировке реализованы в AutoCAD плохо.

При загрузке файла рисунка, изготовленного в AutoCAD R10 или R12, пользователь сталкивается со следующими проблемами:

- вместо текстов на великом и могучем русском языке отображается всякая абракадабра ("зюквы");
- иногда вместо букв рисуются слэши и какие-то цифры;
- становятся нечитаемыми русские имена слоев, блоков и прочих именованных объектов;

- иногда насмерть "падает" система AutoCAD;
- происходят прочие непонятные явления.

Естественно, что причина заключается в том, что старый чертеж выполнен с использованием кодировки, именуемой обычно dos866. Но тут имеются тонкости, которые надо учитывать.

Файлы AutoCAD R10

Если старый чертеж сделан в AutoCAD R10, то отображение старых текстов зависит от наличия "родного" файла шрифта, имеющего DOS-кодировку. Win-AutoCAD, загружая такой файл, ничего не делает с содержимым. Все надписи остаются в кодировке dos866, но их отображение зависит от наличия файла шрифта. Как правило, во всех версиях AutoCAD имеется стандартный набор шрифтов (txt.shx, simplex.shx и т. д.). Имена файлов одинаковые, а кодировки — разные. Если, например, тексты в DOS-файле были выполнены с использованием шрифта txt.shx, то тексты будут отображаться неправильно, потому что Win-AutoCAD имеет файл с таким же именем, но с кодировкой win1251.

Хитрые люди всегда использовали нестандартные имена шрифтов (russ.shx, gost.shx, ar.shx и т. п.). Даже если это была просто копия стандартного файла. Наиболее дальновидные использовали имена типа russ_dos.shx. Теперь они спокойно кладут на пути поиска AutoCAD файлы russ_dos.shx и russ_win.shx и не имеют ряда проблем. Старый чертеж находит файл russ_dos.shx и старые тексты отображаются этим шрифтом, а для новых текстов используется russ_win.shx. Иногда, если файл надо только просмотреть или отпечатать, достаточно и этого, без хитрых манипуляций.

Если импортировать текст, который нормально отображается, в диалоговый редактор строки, то увидим всякие закорючки, и немудрено: текст имеет кодировку dos866, а редактор отображает символы в win1251. Новые надписи будут рисоваться кодировкой win1251 и будут отображаться правильно, если для них установлен стиль, использующий файл с кодировкой win1251, и неправильно, если текущий стиль будет использовать dos866.

Именованные объекты (имена слоев, блоков, видов, видовых экранов, систем координат, типов линий, стилей) в диалоговых окнах Win-AutoCAD будут отображаться неправильно, т. к. они выполнены в dos866, а диалоги выводят символы в win1251.

Все это может свести с ума пользователя, особенно если он не понимает разницы между *стилем* текста и *файлом* шрифта.

Файлы AutoCAD R12

Здесь все еще хуже.

- AutoCAD R12 имел системную переменную DWGCODEPAGE (AutoCAD R12 имел DOS- и Win-версии).
- Глупая в отношении всего "неамериканского" система Win-AutoCAD, загрузив файл с DWGCODEPAGE = dos866, радостно начинает конвертировать имеющиеся в ней тексты. Но делать она этого не умеет, и в результате вместо русских символов появляются символы косой черты с цифрами, соответствующими но-

меру символа в кодовой таблице. Это беда не только AutoCAD, но и некоторых других программ. Глубинный анализ причин такого поведения (UNICODE и пр.) мы рассматривать не будем. Просто констатируем факт.

- Более того, встретив некоторые русские символы в именах, например слоев, система AutoCAD вообще может "рухнуть", "не оставив даже завещания".

Что придумано Autodesk

Прямо скажем, ничего умного.

В поставку системы AutoCAD R14, в составе дополнительных программ Bonus Tools, включали утилиту DBTRANS, расположенную в файле \Bonus\CADTools\dbtrans.arx.

Эту утилиту надо было загрузить, а потом вызвать, набрав в командной строке

```
DBTRANS
```

После запуска этой программы появляется возможность выбора, из какой кодировки, в какую перекодировать рисунок.

Скорее всего, там не будет ни dos866, ни win1251 (нужных нам). Кодовые таблицы описаны в файле dbtrans.cpm. Описания русских кодировок приходилось делать самостоятельно, теперь их можно "достать".

После того как вы раздобудете версию этого файла с русскими кодировками, снова запустите утилиту DBTRANS, укажете в левом окне DOS 866 (Cyrillic), а в правом окне — ANSI 1251 (Microsoft Windows Cyrillic ANSI), и щелкнете по кнопке **Preview** (Просмотр), то в появившемся окошке увидите предварительный результат перекодировки. Очень может быть. Если не произойдет крах системы AutoCAD.

После того, как вы щелкнете по кнопке **OK**, давая разрешение на перекодировку, скорее всего, "рухнет" AutoCAD. Иногда вместе с Windows. Но иногда все происходит как надо, особенно если русских символов в рисунке вообще нет.

После чего ваше уважение к фирме Autodesk будет сильно подорвано.

Происходит это потому, что имеется несколько "уровней", в которых могут встречаться русские буквы. Первый, верхний уровень — тексты, написанные непосредственно в рисунке. С ними и утилита DBTRANS справляется. Далее могут быть русские имена слоев, блоков, а в блоках, в свою очередь, — русские имена вложенных блоков, в них — слои, стили и т. д. Вот это уже не поддается пониманию "фирменных" разработчиков.

Но *наши* люди могут найти выход из любой ситуации. Например, все именованные объекты именуют нерусскими именами. Очень хороший способ, лучше может быть только вообще отказ от русских букв.

Разовое конвертирование можно выполнить путем сохранения файла из DOS-версии AutoCAD в текстовый вариант формата DXF, перекодировки его как обычного текстового файла и последующей загрузки в новую систему AutoCAD.

Однако большинству пользователей требуются программы для перекодировки DWG-файлов. Такие программы довольно часто встречаются в Интернете. Однако абсолютно надежной мы не встречали.

Нам, в свое время, требовалось отконвертировать несколько тысяч файлов. Такую работу выполнить можно только в пакетном режиме, запустив систему AutoCAD на

несколько суток. Естественно, что "ломка" его в этом случае недопустима. Поэтому пришлось исследовать вопрос досконально и придумать свое решение и продумать технологию такой работы, которая заключается в следующем.

- **Шаг 1.** Если предстоит конвертировать файлы системы AutoCAD R12, то необходимо избавиться от упоминания в них о DWGCODEPAGE = dos866. Сделать это можно, открыв рисунок в AutoCAD R12 и установив значение системной переменной DWGCODEPAGE, равное "undefined" (именно так!). Делать это много раз неудобно, да и система AutoCAD R12 может быть недоступной, поэтому лучше воспользоваться утилитой wnewcp.exe, которая делает это без запуска системы AutoCAD.

Замечание

Вопреки распространенному суеверию, утилита WNEWCP не конвертирует рисунок! Хотя там и есть кнопка **Start Conversion** (Начать преобразование). Она только изменяет значение DWGCODEPAGE.

Но делает это без участия AutoCAD и для целого каталога сразу, хотя можно и выбрать отдельный файл. После обработки этой утилитой система AutoCAD не будет пытаться сама отконвертировать рисунок в момент загрузки до запуска LISP-программы конвертирования.

- **Шаг 2.** Надо подумать о судьбе русских имен именованных объектов. Конечно, в идеале, они должны остаться русскими. Но у некоторых "билдов" (вариантов компоновки исполняемого модуля) AutoCAD есть "глюк" (то есть счастливо замеченная вами ошибка), при котором некоторые сочетания русских букв гробят его. Причем установить закономерность не удалось. Это не только "нехорошие" буквы "я", "й", "ю", но и разные сочетания, например: "15Б1п" (марка вентиля и т. п.). Устанавливается это путем экспериментов. Мы это делали только в отдельных случаях. Для чего приходилось делать транслитерацию русских названий. Для выполнения этих операций предназначена программа engtable.lsp, которая:
  - производит транслитерацию русских символов в латинские в именах слоев, блоков, гарнитур, ПСК, видов, видовых экранов, типов линий. При этом, например, МОЙ_ЛЮБИМЫЙ_СЛОЙ будет переименован в MOJI_LYUBIMYJI_SLOJI;
  - для AutoCAD R12 устанавливает системную переменную DWGCODEPAGE в "undefined", после чего даже сам AutoCAD R14 может произвести конвертирование;
  - не производит собственно конвертирования текстов!
- **Шаг 3.** Выполнить конвертирование рисунка. Конвертирование производится с помощью специальной программы конвертирования.

Исходные тексты этих программ мы рассмотрим в *части IV* этой книги. Там же мы разберем, как одновременно с конвертированием произвести и другие массовые операции по обработке файлов, например переименование блоков и слоев для стандартизации их названий, изменение свойств примитивов, изготовление DWF-файлов и т. п.

Утилита Amethist CAD Converter

Эта утилита производит пакетное конвертирование файлов из всех форматов системы AutoCAD от версии R2.5 до R15 "туда и обратно". Программа незаменима в тех случаях, когда требуется изменить формат файлов без участия самой системы AutoCAD. Но изменение самих текстов утилита не производит!

Какие изменения требуется вносить в программы

Помимо очевидной необходимости изменения кодовой страницы текстовых файлов с исходными текстами LISP-программ потребуются более радикальные изменения.

Как учитывать особенности локализованных версий AutoCAD

В локализованных версиях системы AutoCAD можно использовать русские имена команд. AutoCAD R10 мог быть или русским, или английским. В русском AutoCAD R10 применялись только русские имена команд и, соответственно, в программах — только русские названия команд. Начиная с AutoCAD R12, появилась возможность использовать и локализованные, и оригинальные (английские) имена команд. Оригинальное имя команды (и опции команд) начинаются с символа подчеркивания. Разумеется, при программировании необходимо применять оригинальные имена команд — это делает программу независимой от локализации AutoCAD.

Рассмотрим примеры.

Неправильный синтаксис:

- `(command "PLINE" pt1 "w" 0 0 pt2 pt3 "close")`

  Такую инструкцию русифицированная система AutoCAD не выполнит, т. к. она не "знает" команды PLINE, опций **w** и **close**.

- `(command "ПЛИНИЯ" pt1 "Ш" 0 0 pt2 pt3 "З")`

  Такую инструкцию не поймет английская система AutoCAD.

- `(setq ss (ssget "x" '((0 . "ARC") (210 0.0 0.0 -1.0))))`

  Такую инструкцию не поймет русифицированная система AutoCAD (ошибка, широко распространенная среди программистов).

Правильный синтаксис:

- `(command "_.PLINE" pt1 "_w" 0 0 pt2 pt3 "_close")`

  Такую инструкцию правильно поймет и обработает любая система AutoCAD.

- `(setq ss (ssget "_x" '((0 . "ARC") (210 0.0 0.0 -1.0))))`

  Такую инструкцию правильно поймет и обработает *почти* любая система AutoCAD, за исключением безграмотно локализованных.

Замечание

Обратите внимание, для команд префикс (модификатор) "_.", для опций — префикс "_". Точка в префиксе команды означает, что используется "настоящая" команда, а не переопределенная пользователем. Разработчику не следует думать, что если он работает исключительно с англоязычными версиями программ, то так же должны поступать и конечные пользователи.

К сожалению, в некоторых русифицированных версиях AutoCAD "специалисты", проводившие локализацию, умудрились русифицировать то, чего ни в коем случае нельзя переводить:

- имена типов линий в файле acad.lin, например:

```
*осевая,Осевая ____ _ ____ _ ____ _ ____ _ ____ _ ____
A,1.25,-.25,.25,-.25
```

вместо стандартного

```
*CENTER,Center ____ _ ____ _ ____ _ ____ _ ____ _ ____
A,1.25,-.25,.25,-.25
```

или (с переводом отображаемого описания)

```
*CENTER,Осевая ____ _ ____ _ ____ _ ____ _ ____ _ ____
A,1.25,-.25,.25,-.25
```

- символы опций для функции `ssget`, которые пользователь вообще никогда не видит.

Это очень опасные ошибки! Обычный пользователь, работающий с "голой" системой AutoCAD, никогда с ними не столкнется. Такие ошибки выявляются только при работе приложений, к тому же не сразу, а только при обращении к неверно переведенным компонентам. Если излишнее рвение при переводе acad.lin легко нейтрализовать добавлением в него "родных" описаний, то неверные опции `ssget` исправить невозможно. Проявляться такие ошибки могут в виде непредсказуемых фатальных ошибок системы AutoCAD.

Попробуйте потом доказать пользователям, что их легально приобретенную, причем за очень немалые деньги, систему AutoCAD, которая "всегда хорошо работала", вдруг надо заменять "из-за вашей" САПР.

Как использовать Migration Assistance

В составе системы AutoCAD, начиная с версии 2000, поставляется набор утилит AutoCAD Migration Assistance, включающий:

- Layer State Converter — умеренно бесполезный конвертор сохраненных состояний слоев, созданных программой LMAN из Express Tools, в новый формат;
- Batch Drawing Converter — конвертор DWG-файлов между версиями R13, R14 и AutoCAD 200*x*. Для работы требуется сама система AutoCAD. Утилита совершенно бесполезная, т. к. перекодировку символов не производит;
- AutoLISP Compatibility Analyzer — на вид интересный анализатор кода LISP-программ. Выискивает места, сомнительные для действующей версии, и вставля-

ет комментарии, напоминающие о необходимости изменений. Утилиту можно отнести к умеренно полезным, т. к. "натаскана" она на небольшое количество сомнительных ситуаций (применение команды OPEN, использование некоторых системных переменных);

- Menu and Toolbar Porter — бесполезная для разработчиков утилита для "конверсии" файлов меню;
- Command Alias Porter — бесполезная для разработчиков утилита "конверсии" псевдонимов команд.

Использовать Migration Assistance, скорее всего, придется только один раз — для того, чтобы лишний раз убедиться, как далека фирма Autodesk от потребностей *наших* пользователей.

Переход на работу под AutoCAD 2004

Когда эта книга задумывалась, последней версией системы AutoCAD была 2002. Большинство программ, описываемых в данной книге, рассчитаны и проверены на работу именно в этой версии. В марте 2003 года фирма Autodesk выпустила последнюю версию AutoCAD — 2004 (как в СССР, "на год раньше срока"!). Фактически это давно ожидавшаяся "четная"[1] версия 16. Теперь уже встает вопрос о миграции в AutoCAD 2005[2]. В книге мы постараемся уделить достаточно места (на самом деле все программы мы будем писать именно для AutoCAD 2004) особенностям этой версии, пока же сделаем краткий обзор новых возможностей.

Учет особенностей AutoCAD 2004

Версия AutoCAD 2004 имеет ряд особенностей, которые надо учитывать. Из множества новинок мы рассмотрим только те, которые имеют непосредственное отношение к разработке программ.

Новинки AutoCAD 2004

- AutoCAD 2004 более не требует, чтобы пользователь относился к группам Administrators или Power Users. Это непосредственно влияет на концепцию размещения пользовательских данных и настроек, о чем мы говорили ранее.
- Приложения, использующие ObjectARX и ObjectDBX, должны быть, как минимум, перекомпилированы с применением Microsoft Visual C++ 7.0 (часть Visual Studio .NET).
- При использовании ObjectDBX должны быть заменены ссылки на DLL с AxDb15.dll на AxDb16.dll. Обращения к объектам должны быть изменены с `ObjectDBX.AxDbDocument` на `ObjectDBX.AxDbDocument.16`.
- При использовании библиотек типов следует обращаться к AutoCAD 2004 как `AutoCAD.Application.16`.

---

[1] Случайно или намеренно, но именно четные версии AutoCAD (10, 12, 14) были наиболее популярны и стабильны, а нечетные оставляли впечатление экспериментальных и полусырых.

[2] Об учете особенностей версии AutoCAD 2005 *см. главу 38.*

- Visual LISP запоминает свои настройки при работе под Windows 2000/XP в \Documents and Settings\%UserNname%\Local Settings\Application Data\Autodesk\AutoCAD\<language-code>\vlide.dsk.
- В связи с использованием в системе AutoCAD 2004 цветов типа True Color потребуются изменения в функциях, работающих с цветом. Примеры рассмотрены в *части III*.
- Внесены изменения в объект `Preferences`.

Замечание

Далее для имен каталогов мы будем применять подстановку %apppath%, которой в Windows 2000/XP будет соответствовать значение %userprofile%\Application Data\Autodesk\AutoCAD 2004\R16.0\enu, и подстановку %localpath% для %userprofile%\Local Settings\Application Data\Autodesk\AutoCAD 2004\R16.0\enu.

Для объекта `PreferencesFiles` будут возвращаться следующие значения свойств (некоторые, не интересующие нас в данный момент, свойства опущены):

- `AutoSavePath = "%userprofile%\\Local Settings\\Temp\\"`
- `ColorBookPath = "C:\\Program Files\\AutoCAD 2004\\support\\color; %apppath%\\support\\color"`
- `ConfigFile (RO) = "%localpath%\\acad2004.cfg"`
- `CustomDictionary = "%apppath%\\support\\sample.cus"`
- `FontFileMap = "%profilepath%\\support\\acad.fmp"`
- `LogFilePath = "%apppath%\\"`
- `MenuFile = "%apppath%\\support\\acad"`
- `PrinterConfigPath = "%apppath%\\plotters"`
- `PrinterDescPath = "%apppath%\\plotters\\PMP Files"`
- `PrinterStyleSheetPath = "%apppath%\\Plot Styles"`
- `PrintSpoolerPath = "%userprofile%\\Local Settings\\Temp\\"`
- `SupportPath = "%apppath%\\support;C:\\Program Files\\AutoCAD 2004\\support; C:\\Program Files\\AutoCAD 2004\\fonts;C:\\Program Files\\AutoCAD 2004 \\help;C:\\Program Files\\AutoCAD 2004\\express;C:\\Program Files\\ AutoCAD 2004\\support\\color"`
- `TempFilePath = "%userprofile%\\Local Settings\\Temp\\"`
- `TemplateDwgPath = "%localpath%\\Template"`
- `TempXrefPath = "%userprofile%\\LOCALS~1\\Temp\\"`
- `TextureMapPath = "%localpath%\\textures"`
- `ToolPalettePath = "%apppath%\\support\\ToolPalette"`
- `WorkspacePath = "%apppath%\\Data Links"`

Аналогичные изменения коснулись и других объектов, связанных с настройками.

Примечание

Если проанализировать, что делает система AutoCAD 2004 при инсталляции, то можно прийти в ужас. LOG-файл инсталлятора имеет размер более 4 Мбайт, компоненты сис-

темы устанавливаются более чем в сотню папок, включая папки \Common Files\Microsoft Shared\Office10\ и подобные, в реестре регистрируется несколько тысяч компонентов.

- Введены новые системные переменные: CLEANSCREENSTATE, GFANG, GFCLR1, GFCLR2, GFCLRLUM, GFCLRSTATE, GFNAME, GFSHIFT, GRIPHOVER, GRIPOBJLIMIT, GRIPTIPS, HPASSOC, INTERSECTIONCOLOR, INTERSECTIONDISPLAY, LOCALROOTPREFIX, MTEXTFIXED, MTJIGSTRING, MYDOCUMENTSPREFIX, PALETTEOPAQUE, PEDITACCEPT, REPORTERROR, ROAMABLEROOTPREFIX, SIGWARN, STANDARDSVIOLATION, STARTUP, TPSTATE, TRAYICONS, TRAYNOTIFY, TRAYTIMEOUT, TSPACETYPE, XREFNOTIFY.
- Внесены изменения в системные переменные: ACISOUTVER, CECOLOR, CHAMFERA, CHAMFERB, CHAMFERC, DWGCHECK, FILLETRAD, MAXSORT, MIRRTEXT, OBSCUREDCOLOR, SAVETIME, SORTENTS, VIEWRES, ZOOMFACTOR.
- Удалены системные переменные: PLOTID, PLOTTER, STARTUPTODAY.
- Появились новые команды: 3DCONFIG, CLEANSCREENON, CLEANSCREENOFF, HLSETTINGS, JPGOUT, MREDO, PNGOUT, PUBLISH, QNEW, REVCLOUD, SECURITYOPTIONS, SETIDROPHANDLER, SIGVALIDATE, TIFOUT, TOOLPALETTES, TOOLPALETTESCLOSE, TRAYSETTINGS, WIPEOUT, XOPEN.
- Изменены команды: BHATCH, BMPOUT, CHAMFER, CHECKSTANDARDS, COLOR, DDVPOINT, ETRANSMIT, LAYOUTWIZARD, MTEXT, NEW, PAGESETUP, PLOT, SAVE, SAVEAS, STANDARDS, TOOLBAR, QUIT, VIEWRES, WHOHAS, WMFOUT, XATTACH, XREF.
- Удалены команды: DWFOUT, ENDTODAY, MEETNOW, TODAY.
- Панели инструментов "нового образца" описываются в XML-формате. Это дает разработчикам большие возможности по управлению ими и может радикально изменить поход к интерфейсу.

О совместимости приложений

На уровне Visual LISP наши приложения явно будут совместимыми потому, что мы не будем забывать заглядывать в документацию. В каждой новой версии есть свои особенности, но, если программист не использует экзотических приемов (таких как функции `if`, `and`, `cond`, `setq` в "Special forms"), приложения, *как правило*, будут совместимыми.

Могут оказаться несовместимыми программы, обращающиеся к `AutoCAD.Application`, а не к `AutoCAD.Application.16`. Нельзя будет работать с ARX-приложениями, откомпилированными для AutoCAD 2000/2000i/2002.

Нам придется создавать отдельные каталоги для компонентов ruCAD разных версий. С большинством программ обе системы AutoCAD (2002 и 2004) смогут работать и даже одновременно, но, например, откомпилированные меню уже не совместимы. Если файл меню будет откомпилирован в системе AutoCAD 2004, то при последующей загрузке этого меню в систему AutoCAD 2002 она "рухнет". В некоторых программах придется делать отдельные ветви в зависимости от текущей версии AutoCAD.

Не выбрасывайте старую систему AutoCAD

Если вы осуществляете переход с DOS-версии AutoCAD на версию для Windows, ни в коем случае не выбрасывайте старую систему. Сохраните ее хотя бы на одном стареньком компьютере. Многие работы удобнее выполнить именно в старой версии — произвести перекодировку рисунка, переименовать именованные объекты в латинские. Даже если вы полностью завершите миграцию своих файлов, нет никакой гарантии, что этого не придется делать с чужими.

Какие программы требуется разработать нам

Для облегчения работы пользователей нам необходимо включить в состав ruCAD программы для следующих операций:

- конвертирования DWG-файлов из DOS-кодировки в Win-кодировку;
- автоматической "стандартизации" DWG-файлов путем изменения имен блоков и слоев, а также свойств объектов (цвета, стиля текста и т. п.) на стандартные.

Разработкой таких программ мы займемся в *части IV* этой книги, а пока не забудем включить их в свои рабочие планы.

ГЛАВА 5

Итоги части I

Все разработчики начинают новый проект с грандиозных планов. Они клянутся, что уж в *этом-то проекте* будут все тщательно проектировать, абсолютно все и сразу документировать, тщательно выполнять тестирование. Разумеется, для э*того проекта* они составят реальный план-график и будут его неукоснительно выполнять.

Увы! Такие обещания никогда не выполняются, и никогда не будут выполняться, по крайней мере авторам такие отклонения от нормальной человеческой психологии неизвестны.

Мы тоже составили величественные планы и принесли "страшные клятвы". Неужели все это бесполезно? Зачем мы потратили столько места в книге? Не лучше ли было опубликовать сотню-другую "исходников"?

Авторы надеются, что первую часть написали не зря. Конечно, найдется немало читателей, которые скажут, что все это "мура", "избито", "никому не нужно", "делать надо не так, а наоборот". Надеемся, что так будут думать не все.

Мы составили и формальное техническое задание (которое лично вам, читатель, может быть и не нужно). Мы уже знаем, что и как должна делать наша система, хотя пока не знаем, как это реализовать. Но мы уже наметили пути реализации. Базовые принципы, которые мы обдумали и зафиксировали, позволят нам продвигаться в правильном направлении.

Начинающий разработчик теперь знает, какие трудности встретятся на пути, а опытный — может сравнить мысли авторов с собственными и выработать приемлемое для себя решение.

Мы изложили собственный "стандарт" работы с системой AutoCAD. Для того чтобы стандарт действовал, мы наметили, какие программы придется разработать. Возможно, наш "стандарт" пригодится и тем пользователям, которые вовсе не будут работать именно с нашей системой.

Мы продумали систему каталогов для размещения компонентов системы. Теперь, начиная конкретные разработки, мы сразу будем знать, что куда положить.

Замечание

Откроем секрет. Для того чтобы изложить только базовые принципы, авторам уже пришлось разработать более сотни функций, предназначенных для проверки возможности реализации системы. Кое-какие из первоначальных замыслов пришлось отбросить.

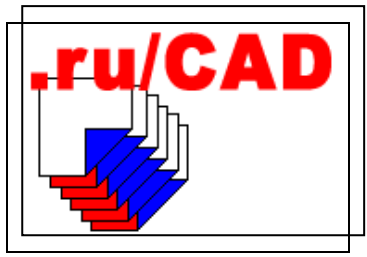
Наконец, мы придумали не самое плохое название для своей системы, и даже сделали логотип. Это не так уж мало. У нас еще нет конечного продукта, а мы уже можем его рекламировать.

Найдутся такие читатели, которые смогут найти лучшие решения при составлении подобного плана. Это вполне естественно: то, что придумал один, кто-то обязательно сможет улучшить — совершенству нет предела. Но в любом деле когда-то надо ставить точку, иначе мы рискуем потратить на "вылизывание" схемы столько сил, что их не останется ни на что другое. Нельзя забывать и то обстоятельство, что любая сложная система — плод множества компромиссов.

В *части I* книги мы говорили о том, *что* надо делать, в следующих частях будем разбирать, *как* это делать.

Итак, вперед!

ЧАСТЬ II



Адаптация AutoCAD без программирования

ГЛАВА 6

Использование блоков

В *части II* мы рассмотрим вопросы, относящиеся к адаптации AutoCAD, т. е. к тому, что можно сделать без программирования. Часто адаптацией занимаются подготовленные пользователи, но на таком уровне усовершенствование системы обычно заканчивается формированием коллекций блоков, типов линий и штриховок, в лучшем случае с включением в меню. Мы постараемся использовать все лучшие возможности адаптации, но сделаем это с максимальными удобствами для пользователей.

Самым эффективным из простых средств повышения производительности "электронного кульмана" является использование блоков. В системе AutoCAD имеется много штатных средств для работы с блоками, и эти средства постоянно совершенствуются. Достаточно вспомнить появившиеся в версии AutoCAD 2000 Центр управления (Design Center) и в версии AutoCAD 2004 — Инструментальные палитры (Tool Palettes). В автоматизируемых нами разделах проектов блоки можно использовать очень широко. Количество применяемых блоков возможно будет исчисляться тысячами, и для работы с ними потребуются эффективные программные средства. Но сначала блоки необходимо приготовить, а для того чтобы делать их максимально полезными, нужно придерживаться определенных правил, которые мы и рассмотрим в этой главе.

Какие виды блоков требуется использовать

Множество используемых блоков можно разделить на несколько видов, каждый из которых предназначен для решения своего круга задач.

Блоки-чертежи

Каждый рисунок, сохраненный в файле, можно вставить в другой рисунок в виде блока, расчленить на составляющие объекты и откорректировать. Это очень эффективное решение, но имеющее некоторые недостатки. Например, во вставляемом рисунке-блоке могут быть устаревшие или неверные технические решения. Управлять такими процессами мы не будем, но дадим пользователям возможность более комфортного выбора вставляемых рисунков.

Частным, но важным вариантом блоков-чертежей являются *типовые проектные решения* (ТПР). В САПР типовые проектные решения могут быть более удобным аналогом бумажных ТПР. Главная их особенность в том, что они должны накапливаться в специальной электронной "библиотеке типовых проектов". Готовиться ТПР должны тщательно, т. к. предполагается их использование разными исполнителями. В библиотеку ТПР могут быть помещены планы жилых секций, часто повторяющиеся узлы, чертежи тепловых пунктов, насосных и т. п. В состав ruCAD мы включим только несколько ТПР, просто для примера использования технологии. Что именно сохранять и использовать в качестве ТПР должны решать специалисты проектной организации, т. к. в проектном деле очень много субъективных моментов и решения, с успехом тиражирующиеся в одной организации, могут оказаться непригодными для другой.

В блоках-чертежах может быть множество слоев и это главное их отличие от других видов блоков.

Блоки-изделия

В таких блоках содержатся изображения различных реальных изделий, выполненные в натуральную величину. Это могут быть и строительные конструкции, и любые виды оборудования, и даже целые здания и типовые сооружения (например, для размещения на генпланах). Блоки этого вида мы всегда будем создавать на одном слое, в натуральном масштабе с единицей рисунка "миллиметры", даже если такие блоки будут вставляться только в рисунки генпланов и наружных сетей.

В составе таких блоков не должно быть нанесенных размеров и надписей (если надпись не является частью самого изделия), но могут быть атрибуты.

Частным случаем блоков-изделий будут *блоки с изменяемой шириной линий*. Многие виды изделий, даже в одном рисунке и на одном слое, требуется изображать или тонкой, или "толстой" линией. Для облегчения этих процессов у нас будут предусмотрены специальные функции, но выделить такие блоки в специальный "подвид" будет разумно.

Единичные блоки

Единичные блоки рисуются в габаритах одной единицы рисунка (неважно, какой). Один единичный блок может заменить тысячи других блоков. Например, блок, нарисованный в виде квадрата размером 1×1 единиц, может вставляться с разными масштабными коэффициентами по осях X и Y. Вставленный единичный блок может изображать и стол, и шкаф, и здание, и шинопровод. Единичный размер используется для удобства задания реальных размеров реальных объектов. Этот прием применяется очень часто.

Менее очевидным является то, что единичные блоки могут быть не только квадратными, а и круглыми, да и вообще любой формы. Например, единичный круг с осевыми линиями может изображать и болтовое отверстие, и люк колодца, и большой резервуар. Единичный отрезок может вставляться и в качестве бергштриха у топографических горизонталей, и изображать стык трубопроводов, и применяться для изображения откосов у насыпей.

Изображение единичного блока не обязательно должно быть простым. Хоть такой блок при создании кажется "маленьким", но вставляться-то он может с любыми масштабными коэффициентами. Важно только, чтобы при масштабировании блока со значительно отличающимися коэффициентами по осям X и Y, конечное изображение имело бы допустимые искажения. Растянутый квадрат будет хорошо "смотреться" всегда, а слишком растянутое изображение электродвигателя может оказаться неузнаваемым.

Единичные блоки могут и не вписываться в единичный квадрат. Например, целый ряд изображений электродвигателей одной серии можно заменить единственным блоком, размером 1×2 единицы. Такой блок будет вставляться с меньшими искажениями (конечно, исходный размер такого блока надо знать программисту, но у нас-то все будет делаться программным путем, пользователю этих деталей знать не нужно).

В строительстве изображения многих изделий носят условный характер, соотношения внутренних размеров элементов изображения изделия не имеют значения, важен только внешний габарит. Например, если мы сделали единичный квадрат с изображением псевдоштриховки древесины поперек волокон, то такой блок прекрасно изобразит и квадратный брус, и доску.

Блоки-символы

Огромное количество условных знаков и изображений можно и нужно использовать в виде блоков. Размеры условных обозначений устанавливаются стандартами в миллиметрах на бумаге. Именно такими размерами и надо создавать подобные блоки, это значительно облегчит их вставку. При этом мы будем использовать только один типоразмер из допускаемых по ГОСТ. Например, если знак высотной отметки по ГОСТ может иметь полочку от 11 до 15 мм, размер стрелки засечки от 2 до 4 мм, высоту от стрелки до полочки от 2 до 6 мм, то мы нарисуем такой блок с размерами полочки 11 единицы, стрелки 2 единицы, высотой 3 единицы и с атрибутом высотой текста 2,5 единицы.

Некоторые блоки-символы одновременно являются и блоками-изделиями. Типичный пример — дорожные знаки. С одной стороны, это типичные символы, которые могут показываться на картах и планах, чтобы размер изображения на бумаге был, например, 8 мм. С другой стороны, дорожный знак может быть реальным изделием, показываемым на установочном чертеже. В этом случае важны его реальные размеры, которые, к тому же, могут быть разными в зависимости от условий установки. Блоки дорожных знаков лучше сделать *единичными*, но с достаточно точной проработкой деталей. Это позволит удобно применять их в различных целях.

Блоки-таблицы

В строительном проектировании используется огромное количество таблиц. Некоторые из них имеют фиксированное количество граф и строк, некоторые — переменное, но большинство таблиц имеет переменное количество строк. Вполне разумно "шапки" таблиц делать в виде блоков (иногда с редактируемыми атрибутами в заголовке), а требуемое число строк рисовать программным путем. Разумеется, мы не будем разрабатывать программы для рисования всех таблиц, достаточно иметь пару универсальных функций.

Какие блоки нам не нужны в системе

В свою систему мы не будем включать некоторые виды блоков, применение которых, казалось бы, может дать самый большой эффект по повышению производительности труда. Прежде всего, это *блоки-чертежи*, о которых было сказано ранее. Помимо чертежей, в виде блоков очень удобно вставлять укрупненные узлы. Например, при проектировании отопления и вентиляции очень удобно вставлять блоками узлы обвязок отопительных приборов и воздухонагревателей, при проектировании электрооборудования — типовые схемы управления электродвигателями, а при разработке технологии производства — планировочные решения. Мы не будем пытаться оказывать влияние на техническую политику проектной организации. Предоставленные ruCAD средства позволят легко наработать библиотеки таких типовых проектных решений, при необходимости мы их также сможем быстро сделать, но вне *поставляемого* комплекта системы.

Как ни странно, но мы не будем создавать и блоки, с которых все начинают — блоки форматов. В строительном проектировании приходится использовать форматы листа из более широкой номенклатуры, чем "джентльменский набор" А1—А4. Часто применяются и длинные узкие листы (трассы и профили коммуникаций) и листы большого размера (генпланы и технологические планировки). О возможности использования нестандартных или "почти стандартных" размеров листа мы уже писали в *части I*. Вместо множества блоков-форматов мы напишем одну программу, которая позволит нарисовать формат любого размера с любой формой основной надписи. А вот сами основные надписи мы сделаем в виде блоков с атрибутами, для того, чтобы графы основных надписей было бы удобно заполнять и редактировать и вручную, и программно, да еще так, чтобы большую часть повторяющихся текстов (наименование стройки и т. п.) приходилось "набивать" вручную только один раз.

Как правильно создавать блоки

Создавать блоки нужно очень тщательно, придерживаясь строгих правил. Один неправильно нарисованный блок может испортить сотни чертежей, в которые он будет вставлен. На работу по рисованию блоков лучше "назначить" не программистов, а обычных пользователей, причем очень дотошных и аккуратных. Блоков придется рисовать тысячи, держать на этих работах специальных "нормоконтролеров" накладно, здесь все должно быть на совести "художников". Самое главное, чтобы блок был нарисован точно по размерам, с использованием объектной привязки, режимов ORTHO и SNAP.

Исполнитель не должен полагаться на свои знания и опыт, необходимо каждое стандартизированное изображение сверить с действующим нормативным документом, а не со справочниками или учебниками. Если в "вашем" стандарте (например, на оформление чертежей водопровода и канализации) нет искомого объекта (например, манометра), то следует разыскать его в "чужих" стандартах (например, по средствам автоматизации), а не придумывать свой, и не делать так, "как всю жизнь рисовали".

Как присваивать имена блокам

Правильные имена блоков имеют огромное значение, прежде всего из-за наличия в системе AutoCAD механизма переопределения блоков. Пока вы работаете только со

своими рисунками, никаких проблем нет, но при обмене файлами неприятности обязательно будут. Представьте, что в *свой* файл, в котором есть блок с "оригинальным" именем ОТМЕТКА (изображающий высотную отметку на плане), вставляется *чужой* файл, в котором есть блок с таким же именем, но изображающий высотную отметку на разрезе. Изображения чужих отметок примут вид ваших, и хорошо еще, если это будет замечено. Если вы просто начнете редактировать этот *чужой* файл, но будете использовать свою программу, вставляющую блок ОТМЕТКА, то уже *ваша* отметка будет иметь неправильный вид. Достаточно взглянуть на популярные сборники программ, чтобы убедиться, как ограничена фантазия авторов в именовании блоков (A1, GAIKA, BOLT, OTMETKA, RECTANGLE и т. п.).

Все имена наших блоков должны быть в определенной мере уникальными. В определенной мере, а не абсолютно уникальными потому, чтобы не создавать себе и пользователям слишком больших неудобств при работе с блоками (мы ведь могли бы задумать и реализовать действительно уникальные имена в виде строкового представления GUID (Globally Unique Identifier) типа F8F43F81-6CC6-11D7-8CF1-CA906998697E). Определенная уникальность имен наших блоков будет достигаться добавлением к каждому имени блока префикса "RU_".

Хотя мы постараемся полностью исключить "контакт" обычных пользователей с именами блоков, самим разработчикам придется иметь с ними много дел, поэтому имена блоков должны быть осмысленными. "Осмысленность" имени понятие относительное. Для одного специалиста А510501 (обозначение вентилятора) или 15Б1БК (марка вентиля) говорит очень много, для другого это просто набор символов. Но, поскольку с именами будут работать именно специалисты, подобная "специализация" является допустимой. Можно применять и очень понятные русские имена блоков, но лучше все-таки использовать имена с латинскими символами, причем не с транслитерацией русских слов, а "настоящие" английские слова (ELEVATION, BATH или VALVE, а не ОТМЕТКА, VANNA или VENTIL). И наконец, длину имени блока следует ограничить 31 символом (с учетом возможного использования наших рисунков в устаревших версиях AutoCAD), не применять в них пробелов и символов, недопустимых для имен файлов и для имен в AutoCAD.

Как устанавливать цвет и слой примитивам, входящим в блок

Все примитивы, входящие в определение блоков, должны создаваться на слое 0 и должны иметь цвет BYBLOCK (ПОБЛОКУ), за исключением случаев, когда цвет объектов должен быть всегда строго заданным (например, у дорожных знаков). Цвет BYBLOCK — более гибкий механизм, чем установка цвета примитивов, входящих в блок, со значением BYLAYER (ПОСЛОЮ), т. к. позволяет менять цвет вставки блока отдельно от цвета всего слоя.

Слой 0 — специальный слой системы AutoCAD, предназначенный именно для создания блоков. При вставке блока примитивы, созданные на слое 0, оказываются на текущем слое, а не "где попало".

Как назначать ширину, тип и вес линий

Для примитивов, которые всегда должны быть нарисованы тонкими линиями, следует явно установить вес (lineweight) равным 0. У примитивов, которые всегда долж-

ны иметь фиксированную ширину линий, необходимо задать соответствующее значение веса. Для остальных примитивов в качестве значения веса следует устанавливать BYBLOCK (ПОБЛОКУ).

В блоках, у которых планируется программное изменение физической ширины линий, объекты с изменяемой шириной следует рисовать полилиниями с такой шириной, какая требуется при выводе чертежа на бумагу.

Как устанавливать точку вставки блока

Точку вставки блока при его создании следует выбирать очень тщательно. Точка вставки должна находиться в том месте, которое является наиболее "технологичным" при использовании. Например, у блока высотной отметки для разрезов точка вставки должна быть в вершине "стрелки", у блока высотной отметки для планов — в центре прямоугольника-рамки, у блоков насосов и вентиляторов — на пересечениях осей валов и "улиток".

Следует использовать единую методику для создания блоков, устанавливаемых на различных линиях (трубопроводах и проводниках). Такие блоки необходимо рисовать так, чтобы направление движения среды через изделие, изображаемое блоком, всегда совпадало с осью X. Точку вставки для блоков, устанавливаемых на конце линии, следует принимать в начале блока, у блоков, устанавливаемых на середине линии — в центре блока (это облегчит программную "врезку" блоков в линии).

Создавать блоки и задавать точку вставки рекомендуется при установленной мировой системе координат.

Как правильно включать в блоки атрибуты

До создания блока с атрибутами следует создать определения атрибутов с помощью команды ATTDEF.

При определении атрибутов блоков необходимо правильно задавать их имена. Рекомендуем в диалоговом окне команды ATTDEF применять латинские имена атрибутов (в поле **Tag**), русские подсказки (в поле **Prompt**), а также устанавливать значения атрибутов по умолчанию (в поле **Value**). Текстовый стиль и высоту текстов, в основном, лучше принимать по "нашему" стандарту *(см. главу 4)*. При назначении выравнивания текста следует предусмотреть возможные варианты длины значений атрибутов.

Для того чтобы в диалоговом окне задания значений атрибутов, появляющемся при вставке блока командой INSERT, или в диалоговом окне редактирования значений атрибутов при выполнении команды ATTEDIT атрибуты выводились в необходимом порядке, следует точно в таком же порядке указывать определения атрибутов при формировании набора примитивов, входящих в определение блока.

Как хранить блоки в каталогах ruCAD

Блоков будет очень много, поэтому придумаем схему для их размещения в каталогах системы ruCAD.

Блоки мы будем хранить в каталоге %ruCADRootDir%\All Users и его подкаталогах. Внутри каталога All Users мы создадим следующие подкаталоги:

- Block — для блоков-изделий, сохраняемых в отдельных файлах;
- Block-Lib — для библиотек блоков, внутри которых определено много маленьких единичных блоков и блоков-символов по определенной тематике;
- Block-Lw — для блоков с изменяемой шириной линий;
- Table — для блоков шапок таблиц и их описаний.

Внутри этих каталогов, при необходимости, будем создавать тематические подкаталоги.

Как формировать библиотеки блоков

Рано или поздно у нас появится на диске неимоверное количество маленьких блоков. Каждый из них будет иметь размер 10—25 Кбайт, причем основной объем будет приходиться не на само изображение, а на общую информацию, сохраняемую в DWG-файле.

С учетом особенностей той или иной файловой системы, фактически занимаемое файлами место всегда больше суммы байтов отдельных файлов. Каждый файл занимает целое количество кластеров. В файловой системе FAT16 в зависимости от размера диска размер кластера мог быть до 64 Кбайт, т. е. 1 000 файлов по 1 Кбайту занимали на диске не 1 000 Кбайт, а 64 000. С переходом на современные файловые системы проблема с неполными кластерами потеряла остроту, но дискового пространства всегда недостаточно. Большее значение, чем стоимость жестких дисков, имеет существенное замедление работы различных файловых утилит при обработке большого количества файлов.

Для уменьшения количества файлов применяют так называемые библиотеки блоков. Библиотека блоков — это обычный DWG-файл, в котором обязательно имеются определения нескольких десятков или сотен блоков. Могут быть не только определения, но и вхождения этих блоков.

Примечание

Например, в одной библиотеке блоков имеется 168 маленьких блочков. Размер файла библиотеки 158 Кбайт, в то время как сумма размеров файлов блоков (когда они лежат сами по себе) составляет 5,32 Мбайт!

Для того чтобы использовать блок из библиотеки, необходимо начать операцию вставки библиотечного блока в текущий рисунок. В процессе вставки библиотечного блока, в момент запроса точки вставки, нужно прервать выполнение команды. После этого все определения блоков из библиотеки появятся в текущем рисунке, но самой вставки библиотеки (т. е. вхождения библиотечного блока) не будет. Решать эту задачу программным путем мы будем в *части III*.

Формирование библиотек блоков следует производить разумно. В библиотеках надлежит группировать блоки одинаковой тематики, добиваясь более высокой вероятности одновременного использования большей части элементов библиотеки. Блоки в рисунке библиотеки лучше размещать так, чтобы их удобнее было просматривать и, при необходимости, переопределять.

Какие программы потребуются для работы с блоками

С блоками придется работать очень много, поэтому в число первоочередных программ ruCAD мы включим инструменты для облегчения собственной работы. По опыту прежних разработок нам потребуются служебные программы, выполняющие следующие операции:

- вставку всех отобранных DWG-файлов в рисунок;
- выгрузку всех блоков в отдельные файлы;
- вывод перечня блоков, определенных в рисунке;
- запись перечня блоков, определенных в рисунке, в текстовый файл;
- расстановку всех блоков в виде матрицы с заданным шагом;
- переопределение указанного блока с возможностью изменения набора примитивов;
- задание для вставленных блоков единого масштаба по всем осям;
- переименование для группы файлов слоев и блоков;
- пакетную смену свойств примитивов, входящих в блоки.

ГЛАВА 7

Использование штриховок и типов линий

Что можно усовершенствовать для облегчения работы со штриховками и типами линий?

Средства для работы со штриховками

Средств для работы со штриховками в системе AutoCAD вполне достаточно. Образцов штриховок тоже имеется немало, но, как обычно, того, что требуется именно вам, по случайности не оказывается. В строительном проектировании вообще-то используется не так уж много образцов штриховок, они больше нужны для топографических планов, генеральных планов и архитектурных чертежей. В обычных конструкторских чертежах могут с успехом использоваться стандартные образцы. Для некоторых специализированных приложений (например, геологических планов и разрезов) могут потребоваться специальные образцы.

В настоящее время нет никакого смысла тратить время на создание собственных образцов штриховок. Все уже создано! Потратьте несколько часов на поиск в Интернете и вы получите в свое распоряжение сотни образцов. Гораздо больше времени понадобится, чтобы разобраться с этим богатством и выбросить все ненужное. Постарайтесь избежать искушения собрать все найденное в файл acadiso.pat. Большинство образцов вам никогда не понадобится, а загрузка огромной коллекции займет очень много времени.

В ранних версиях системы AutoCAD просматривать образцы штриховок можно было только через слайды. Надо было создать рисунок с примерами штриховок, наделать кучу слайдов, собрать из них библиотеку слайдов, описать каждый слайд в разделе IMAGE файла меню и только после этого штриховки более или менее удобно можно было применять. Команда BHATCH использовала слайды образцов штриховок из библиотеки acad.slb, которую надо было дополнять самостоятельно. В современных версиях системы AutoCAD миниатюры для просмотра образцов штриховок генерируются автоматически. Это в целом лучше, но реализовано очень плохо. Миниатюра в диалоговом окне **Boundary Hatch** (Штриховка по контуру) очень маленького размера, хотя поле диалога используется очень нерационально. Коллекция миниатюр в диалоговом окне **Hatch Pattern Palette** (Палитра образцов штриховки) выглядит лишь чуть-чуть лучше. Увы, программисты фирмы Autodesk иногда плохо реализуют хорошие идеи.

Для отбора образцов штриховок лучше создать специальный рисунок. В нем заштрихуйте заготовленные прямоугольники с соотношением сторон 1:1.5 имеющимися образцами, проставьте возле каждого имя штриховки и пометьте (например, цветом) образцы, которые будете использовать.

Отобранные штриховки лучше всего собрать в файл

```
%ruCADRootDir%\Local Settings\Application Data\ruCAD\AutoCAD\acadiso.pat
```

Непременно укажите в комментариях, откуда получен образец и кто его автор.

В процессе отбора вы сразу наметите программы, которые понадобятся для работы со штриховками. Стандартный диалог хорош, когда изредка надо создать одну-две штриховки, но при серьезной работе стандартная команда становится неудобной.

Опыт подсказывает, что потребуется несколько вариантов функции "быстрой штриховки", при использовании которой пользователь не перебирает образцы и не экспериментирует с масштабом штриховки, а выбирает из иллюстрированного меню "разрез ЖБК", "болото" и любое подобное название и быстро выполняет работу.

Очень удобна программа быстрого изменения масштаба штриховки "Гуще-реже". Для того чтобы система знала заранее, в каком масштабе рисовать штриховки, удобно создать специальный файл hatch.ini (листинг 7.1), в котором (с учетом того, что штриховки могут быть добыты из разных источников) для каждого образца сохраняется оптимальный масштаб штриховки.

Листинг 7.1. Фрагмент файла hatch.ini

```
[HatchScale]
ansi31=20.000
GLINA=0.050
LUG=4.000
ANSI37=20.000
ACAD_ISO15W100=0.500
ACAD_ISO14W100=0.500
ACAD_ISO13W100=0.500
WOOD8=8.000
```

Разумеется, для создания такого файла потребуется специальная программа.

Средства для работы с типами линий

Положение с типами линий примерно такое же, как и со штриховками. Все стандартные типы линий (в том числе по стандартам ЕСКД и СПДС) давно разработаны. Собрать коллекцию описаний типов линий очень просто — в Интернете есть все, за исключением того, что вам нужно. С обычными типами линий ("морзянкой") проблем не будет. Специальные типы линий, применяемые в топографии, возможно, придется разрабатывать самостоятельно. Для собственных типов линий лучше для имен задавать "фирменный" префикс. В нашей системе, по традиции, мы будем использовать префикс RU_.

Использовать линии различных типов в AutoCAD достаточно удобно, если определение типа уже имеется в рисунке. Загрузка отсутствующих определений достаточно

проста, но и от этого пользователя можно избавить, если предусмотреть, когда нужно, автоматическую подгрузку требуемого описания.

Создавать новые типы линий лучше всего с использованием программы mkltype.lsp из библиотеки Express Tools. Пользоваться этой программой очень просто.

1. Предварительно нарисуйте прототип будущей линии. При этом лучше установить системной переменной LTSCALE значение 1, а "линию" рисовать с такими размерами, которые должны быть на бумаге. Следует использовать только примитивы LINE, TEXT и SHAPE. Прототип линии должен начинаться и заканчиваться отрезками.
2. Введите команду MKLTYPE (основная функция в файле mkltype.lsp оформлена как команда системы AutoCAD). Программа в диалоговом режиме запросит имя файла определений типов линий, имя создаваемого типа линии и ее описание. Если тип линии с таким именем уже есть в файле, будет выдан запрос на переопределение типа.
3. Далее последуют запросы:
   - ***Specify starting point for line definition:***
     (На этот запрос укажите начальную точку прототипа, используя объектную привязку)
   - ***Specify ending point for line definition:***
     (Укажите конечную точку прототипа, используя объектную привязку)
   - ***Select objects:***
     (Выберите примитивы, входящие в определение, лучше секущей рамкой)
4. Программа сгенерирует описание типа линии, запишет его в заданный файл (другие определения не будут утеряны) и загрузит новый тип линии.

С первого раза может не все получиться, но после нескольких тренировок вы научитесь создавать самые сложные типы линий.

Как создавать "лохматые" линии

К сожалению, только в системе AutoCAD R14 появилась возможность создания так называемых "лохматых" линий, которые, в отличие от традиционной "морзянки", могут иметь выступающие за трассу элементы. Такие линии широко применяются в топографических условных знаках (кабели, ограждения и т. п.). В описание этих типов линий, кроме отрезков, можно включать текстовые символы или формы (примитивы SHAPE). Примеры таких линий имеются в стандартном файле acadiso.lin.

Создавать такие линии удобнее всего с помощью упоминавшейся команды MKLTYPE.

К большому сожалению, в системе AutoCAD отсутствует возможность создания типов линий (и штриховок) с использованием блоков.

Если в определении типа линии используются примитивы формы, то SHX-файл с этими формами должен находиться на доступном пути. Это вообще самый большой недостаток использования форм. Нет никакой гарантии, что вместе с чужим рисунком к вам попадут примененные в нем файлы форм. Хотя в AutoCAD имеется очень полезная команда ETRANSMIT, позволяющая создать архив со всеми требуемыми

файлами (шрифты, формы и другие файлы), но если этой командой можно *не воспользоваться*, то так часто и происходит.

Избежать этих неприятностей можно путем использования символов из обычного файла txt.shx. С помощью различных комбинаций обычных текстовых символов можно получить любой требуемый эффект. Файл txt.shx имеется в каждом комплекте системы AutoCAD, поэтому проблем с совместимостью не будет. Для использования в определениях линий необходимо создать специальный стиль текста RU_LINE, с файлом шрифта txt.shx. На рис. 7.1 мы приводим некоторые примеры типов линий, использующих текстовые символы.

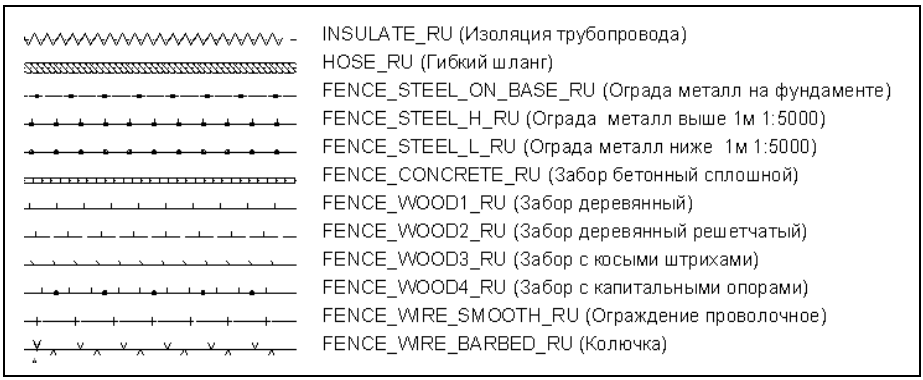


Рис. 7.1. Пример 1 типов линий с текстовыми символами

Описания к этим типам линий приведены в листинге 7.2. Определения остальных типов линий вы найдете в файле ruCAD.lin на компакт-диске.

Листинг 7.2. Описания типов линий к рис. 7.1

```
*INSULATE_RU,Изоляция трубопровода
A,0.1,["Z",RU_LINE,S=2,Y=-1,R=60],-2.2
*HOSE_RU,Гибкий шланг
A,0.1,["S",RU_LINE,S=2,Y=-1,R=0],-1
*FENCE_STEEL_ON_BASE_RU,Ограда металлическая на фундаменте
A,1.5,-.5,.916667,["o",RU_LINE,y=-.333333,s=1],.04,["o",RU_LINE,
y=-.293333,s=.88],.073333,["o",RU_LINE,y=-.22,s=.66],.053333,["o",
RU_LINE,y=-.166667,s=.5],.056667,["o",RU_LINE,y=-.11,s=.33],1.11,-.5,1.5
*FENCE_STEEL_H_RU,Ограда металлическая выше 1 м 1:5000
A,1.916667,["o",RU_LINE,y=-.333333,s=1],.04,["o",RU_LINE,y=-.293333,
s=.88],.073333,["o",RU_LINE,y=-.22,s=.66],.053333,["o",RU_LINE,y=-.166667,
s=.5],.056667,["o",RU_LINE,y=-.11,s=.33],.11,["|",RU_LINE,s=1],2.25
*FENCE_STEEL_L_RU,Ограда металлическая ниже 1 м 1:5000
A,1.916667,["o",RU_LINE,y=-.333333,s=1],.00035,-.222983,["o",RU_LINE,
y=-.11,s=.33],-.06,["o",RU_LINE,y=-.05,s=.15],-.382782,1.917218
*FENCE_CONCRETE_RU,Забор бетонный сплошной
A,0,-.011995,["=",RU_LINE,y=-1.250102,s=2.5],-.006486,.057717,
-1.519896,["|",RU_LINE,y=-.417932,s=.8],-.056835
*FENCE_WOOD1_RU,Забор деревянный
A,2.5,["|",RU_LINE,S=0.8,R=0.0,X=0.0,Y=0.0],2.5
```

```
*FENCE_WOOD2_RU,Забор деревянный решетчатый
A,2.5,["|",RU_LINE,S=0.8,R=0.0,X=0.0,Y=0.0],2.5,-1
*FENCE_WOOD3_RU,Забор с косыми штрихами
A,2,["|",RU_LINE,s=.8,r=45],2
*FENCE_WOOD4_RU,Забор с капитальными опорами
A,2,["|",RU_LINE,s=.8],2.166667,["o",RU_LINE,y=-.333333,s=1],.04,
["o",RU_LINE,y=-.293333,s=.88],.073333,["o",RU_LINE,y=-.22,s=.66],
.053333,["o",RU_LINE,y=-.166667,s=.5],.056667,["o",RU_LINE,
y=-.11,s=.33],2.61,["|",RU_LINE,s=.8],2
*FENCE_WIRE_SMOOTH_RU,Ограждение проволочное
A,2.5,-1,["+",RU_LINE,S=2,R=0.0,X=-0.8,Y=-1],-1,2.5
*FENCE_WIRE_BARBED_RU,Колючка
A,2,["V",RU_LINE,S=1,R=0.0,X=0.0,Y=0.0],4,["V",RU_LINE,S=1,R=180.0,X=0.0,Y=0.0],2
```

Как создавать линии с текстовыми символами

В строительном проектировании широко используются буквенно-цифровые обозначения трубопроводов и инженерных коммуникаций. Такие обозначения указываются на полке выносной линии (вместе с диаметром) или в разрыве линии. Обозначений очень много, но строятся они по единой схеме. Мы не будем для них создавать определения типов линий, а разработаем специальную программу (листинг 7.3), которая будет автоматически добавлять в LIN-файл новые определения. Такая же программа будет создавать типы линий для топографических горизонталей, у которых будут рисоваться бергштрихи, показывающие направление склона, и величина высотной отметки горизонтали.

Листинг 7.3. Примеры автоматически создаваемых типов линий

```
;; RU_TOPO auto-created line
*109,---109---
A,0.25,-0.05,["109", RU_LINE,S=0.1,R=0.0,X=-0.0,Y=-0.05],-0.3,0.5,
["|", RU_LINE,S=0.08,R=0.0,X=0.0,Y=-0.00],0.25
;; RU_TOPO auto-created line
*109_5,---109.5---
A,0.25,-0.05,["109.5", RU_LINE,S=0.1,R=0.0,X=-0.0,Y=-0.05],-0.5,0.5,
["|", RU_LINE,S=0.08,R=0.0,X=0.0,Y=-0.00],0.25
;; RU_CAD auto-created line
*T1,___T1___
A,1,-0.05,["T1", RU_LINE,S=0.1,R=0.0,X=-0.0,Y=-0.05],-0.2,1
```

Особенности использования специальных типов линий

Специальными типами линий пользоваться надо осмотрительно. При неправильном масштабировании линии она может превратиться в сплошную, причем визуально заметить это трудно. В результате в рисунке могут исчезнуть сварные швы, линии грунта и прочие детали.

При рисовании трубопроводов применение типа линии с буквенно-цифровым обозначением не всегда удобно, потому что обозначение живет по своим законам, может оказаться не там, где надо, текст может быть написан "вверх ногами".

Для повышения гибкости системы мы разработаем программы, которые будут, на выбор пользователя, рисовать трассы или специальным типом линии, или вставлять тексты в разрыв обычных линий.

Какие программы потребуются для работы с типами линий

Для облегчения работы с типами линии наша система должна:

- полностью избавить пользователя от необходимости загружать типы линий;
- автоматически создавать определения типов линий для буквенно-цифровых обозначений;
- дать возможность легко изменять направление линий и масштаб типов линий.

ГЛАВА 8

Интерфейс пользователя для работы с AutoCAD

Большое значение для успешной и производительной работы имеет интерфейс системы, его способность изменяться и настраиваться под нужды пользователей.

Что хорошо в стандартном интерфейсе AutoCAD

Система AutoCAD имеет великолепный интерфейс! Оценить его в полной мере можно только поработав в какой-нибудь другой графической системе — сразу почувствуете себя "как без рук".

Командная строка

Самым восхитительным (и незаметным) элементом интерфейса является командная строка. В командную строку выводятся сообщения и приглашения команд, с помощью командной строки пользователь может вводить выражения языка LISP, имена команд и их опций. А как разумно организован ввод данных — в ответ на запрос программы, ожидающей, например, ввода точки, пользователь может и указать точные координаты, и указать смещение по осям X и Y, и задать смещение на заданное расстояние под заданным углом, и ввести с клавиатуры (или выбрать из меню) опцию, полностью изменяющую поведение команды.

Именно командная строка открыла дорогу для разработки приложений тысячам инженеров, которые, зная порядок ввода данных для команды (а ввод еще и протоколируется) и применяя функции `command` и `vl-cmdf` языка LISP, смогли легко разрабатывать программы для решения прикладных задач.

Кроме того, командная строка (точнее, зона командных строк) может приобретать форму окна с названием **Command Line**.

Меню

Меню AutoCAD имеет самое важное достоинство по сравнению со многими системами — оно очень легко поддается адаптации. Разработчику доступно почти все, за

исключением нескольких "зашитых" в AutoCAD элементов управления, обозначенных _Control в описаниях панелей инструментов (их можно только удалить или поменять местами). Можно настроить работу устройства указания (разделы BUTTONSn и AUXn, где n = 1—4), переработать контекстное меню (разделы POP0, POP500—POP999), падающие меню (разделы POP1—POP499), панели инструментов (раздел TOOLBARS), графические меню (раздел IMAGE), экранное меню (раздел SCREEN), планшетные меню (разделы TABLETn, где n = 1—4), строки подсказок (раздел HELPSTRINGS) и горячие клавиши (раздел ACCELERATORS). В AutoCAD можно загружать не только стандартное, но и собственное меню, в том числе "на лету" вставлять в текущее меню разделы из другого файла. Всеми этими действиями можно управлять программно.

Недостаток меню AutoCAD, которое выглядит так же, как меню других приложений Windows, в том, что такое меню плохо приспособлено к специфике САПР. Более конкретно недостатки меню и пути их устранения мы рассмотрим далее.

Панели инструментов

Панели инструментов AutoCAD являются очень удобным средством для быстрого вызова команд. Недостаток их заключается в том, что команд очень много, а места на экране мало. Только в базовой системе AutoCAD около 400 команд. Кроме того, каждый пакет утилит добавляет один-два десятка команд, да еще мы вознамерились добавить несколько тысяч. К счастью, ненужные панели можно легко скрывать и вновь показывать при необходимости.

В системе AutoCAD 2004, по нашему субъективному мнению, разработчики в погоне за "красивостью в стиле XP" перешли границу, за которой "красота" оборачивается неудобствами для пользователя. Кнопки панелей инструментов в AutoCAD 2004 стали хуже читаться, все изображения стали более мягкими, как бы размытыми. Увы, в нарядах от "мастеров высокой моды" не очень-то удобно заниматься повседневной производственной деятельностью.

Центр управления

Это превосходный инструмент, впервые появившийся в AutoCAD 2000. Для пользователей категорий "чайник" или "тетка" Центр управления (Design Center), может быть, и не нужен, однако при работе с "голой" системой AutoCAD и при некоторой подготовке он может оказаться незаменимым.

Мы, при разработке системы ruCAD, будем интенсивно использовать Design Center. Манипуляции с тысячами блоков и формирование библиотек блоков из разных источников будут значительно ускорены. Теперь уже не придется, как несколько лет назад, открывать множество рисунков и разбираться, что же в них имеется полезного.

Tool Palettes в AutoCAD 2004

В AutoCAD 2004 появилось новое средство — Tool Palettes (Инструментальные палитры). Панель **Tool Palettes** имеет несколько логических вкладок (палитр), которые легко может создавать пользователь. На каждую вкладку из Design Center можно

перетащить блоки или образцы штриховок и заливок, при этом для каждого инструмента, размещенного на вкладке, создается миниатюра (растровый образец) для просмотра изображения[1].

Размером миниатюры, в определенных пределах, можно управлять. Блоки можно перетаскивать из любого файла рисунка, образцы штриховок — из PAT-файлов (почему-то некоторые образцы не удается перетащить). Для каждого инструмента палитры можно настроить свойства, в том числе параметры вставки по умолчанию. Использовать инструменты палитры очень просто — достаточно перетащить нужный инструмент в рисунок. При этом при вставке сохраняется возможность как пользоваться значениями параметров по умолчанию, так и задавать свои собственные значения.

Замечание

При использовании Tool Palettes сразу был обнаружен "глюк" — иногда (без видимой закономерности), после перетаскивания блока из палитры, при попытке выбора любого пункта из главного меню AutoCAD появляется сообщение:

Command still active

Фирма Autodesk считает, что проблема возникает из-за того, что пользователь одновременно жмет левую и среднюю кнопку при перетаскивании блока из палитры. Для снятия блокировки они предлагают сохранить рисунок набором команды SAVE в командной строке. По поводу команды SAVE все верно, а вот с причинами они явно темнят.

Описания палитр хранятся в файле

`%AcadUserLocalAppData%\Support\ToolPalette\AcTpCatalog.atc`[2]

Этот файл имеет формат XML[3] и в нем описаны все вкладки, входящие в панель палитр инструментов.

Каждая вкладка сохраняется в подкаталоге Palettes и описывается в аналогичном ATC-файле, на который указывает атрибут `href` в файле AcTpCatalog.atc. Каждый инструмент имеет описание свойств. Иллюстрации к инструментам сохраняются в подкаталоге Palettes\Images в файлах формата PNG размером до 64×64 пиксела. Любую вкладку панели инструментов можно экспортировать в файл формата XML с расширением xtp и импортировать из такого же файла. При экспорте вкладки (палитры) в подкаталоге %DependentFilesPath%\Images сохраняются и файлы иллюстраций.

В целом Tool Palettes можно оценить как очень мощный инструмент. С точки зрения разработчика особенно привлекательным является использование формата XML для хранения настроек. Это открывает огромные возможности по программному управлению палитрами инструментов.

---

[1] В AutoCAD 2005 появилась возможность добавления инструментов с макросами, аналогичными макросам меню.

[2] О псевдонимах системных папок, таких как %AcadUserLocalAppData%, *см. главу 4.*

[3] О формате XML *см. часть III.*

Что плохо в стандартном интерфейсе AutoCAD

Система AutoCAD имеет ужасный интерфейс! Именно так, хотя множество пользователей с нами не согласятся, да и сами мы в начале главы утверждали, что интерфейс AutoCAD великолепен. Противоречие заключается в том, что интерфейс AutoCAD интересен "продвинутым" пользователям, которые любят "поковыряться" в сложных диалоговых окнах со множеством закладок и опций, но стал слишком сложен для обычных пользователей, которым некогда разбираться в этих премудростях, а надо "гнать листаж, зарабатывать деньги и стаж".

Многие новые нужные средства очень плохо реализованы. Например, в раскрывающемся списке слоев обычно не вмещаются полностью не только длинные имена слоев, но и традиционные 30-символьные (в версии AutoCAD 2004 уже лучше: в панели **Layers** (Слои) видно по 25—35 символов, в зависимости от их ширины). Чтобы выполнить простую операцию по отключению слоя, на котором расположен ненужный объект, надо знать имя этого слоя, выбрать этот слой из длинного списка (задерживаясь на каждом имени до появления всплывающей подсказки с полным именем), щелкнуть по "лампочке", да еще не промахнуться, да еще не сделав случайно этот слой текущим.

Таких примеров можно приводить очень много. В целом можно сказать, что интерфейс AutoCAD неудобен для выполнения простых, часто повторяющихся операций. Недаром упрощением многих действий занимаются почти все разработчики, а программы такого характера непременно входят в состав пакетов утилит (Express Tools, ToolPack и т. п.). Сама фирма Autodesk из подобных программ "по просьбам трудящихся" включила в AutoCAD, кажется, только установку слоя по объекту-образцу.

Как мы будем писать главное меню ruCAD

В любой солидной системе должно быть свое собственное меню, хотя фирма Autodesk так не считает. Она рекомендует создавать дополнительные разделы в главном меню AutoCAD, загружая их из отдельных файлов. На самом начальном этапе разработки, пока у нас еще мало работающих программ, мы так и будем поступать, но в дальнейшем, когда "проход" к сотням наших пунктов "через узенькое горлышко" станет слишком утомительным, мы сделаем собственное "большое" меню.

Разрабатывая меню, мы уже начинаем реализацию концепции ruCAD, которую обдумывали в *части I*. Меню является каркасом системы, в нем должны быть найдены места для всех команд, которые мы намечали к реализации. Даже если мы пока не имеем реализации команды, место для нее в меню мы должны предусмотреть. Пусть макроопределения меню вообще не будет, или в нем будет стоять программная заглушка, все равно это будет способствовать получению представления о том, что в конце концов будет.

Какое хотелось бы иметь меню

Autodesk несколько раз меняла концепцию главного меню AutoCAD. Место на критику мы не будем тратить, просто сделаем *свое* меню *как надо*.

За основу мы возьмем стандартный файл acad.mnu, скопируем его в каталог %LocalAcadAllVersionDir% под именем ruCAD.mnu и там будем корректировать. При подключении разных версий AutoCAD (2002 и 2004) файл меню нужно будет копировать в каталоги %LocalAcadAllVersionDir%\15 и %LocalAcadAllVersionDir%\16, т. к. откомпилированные меню версий 15 (2002) и 16 (2004) несовместимы. Разумеется, мы постараемся сделать исходный текст совместимым со всеми версиями.

Редактировать мы будем локализованное меню, т. к., в конечном счете, мы создаем систему для "теток", а не для "профи".

Типы файлов меню

В системе AutoCAD имеется четыре одноименных файла меню с разными расширениями:

- Текстовый файл с расширением mnu. Это *самый исходный* файл, который пишет и редактирует разработчик. В документации по системе AutoCAD 2002 такие файлы еще именовались menu templates (шаблоны меню), в документации по AutoCAD 2004 — original ASCII menu files (исходные ASCII-файлы меню), что является более точным определением.
- Текстовый файл с расширением mns. "Исходный" файл меню (по терминологии AutoCAD). На самом деле он совсем не исходный, а полученный первоначально автоматически из MNU-файла путем копирования, за исключением начального комментария, указывающего на авторские права разработчиков, и других строк комментариев, начинающихся с символов //. В дальнейшем система AutoCAD автоматически отражает в MNS-файле изменения, связанные с действиями пользователя по адаптации панелей инструментов.
- Двоичный файл с расширением mnc. Это компилированный файл, загружаемый в AutoCAD. Создается автоматически и перекомпилируется при изменениях MNS-файла.
- Двоичный файл с расширением mnr. В этом файле, в виде ресурсов, содержатся растровые картинки панелей инструментов.

Кроме того, могут быть в наличии файлы с расширением mnt — двоичный файл ресурсов, создающийся, если MNR-файл защищен от записи, и текстовый файл с расширением mnl. MNL-файл — это сборник LISP-функций, которые должны быть доступны или выполняться при загрузке одноименного меню.

Разработчик должен работать только с MNU-файлом. Так как корректировок будет очень много, а при диалоговой перегрузке надо каждый раз выбирать именно MNU-файл (а не подсовываемый по умолчанию MNC-файл), и не перепутать его с MNS-файлом, да ответить на предупреждение AutoCAD об опасности загрузки MNU-файла, то для ускорения работы просто необходимо написать простенькую программу.

Какие изменения будем вносить в стандартное меню

Сначала мы рассмотрим необходимые изменения на уровне структуры меню, а потом разберем технические детали, связанные с написанием пунктов меню.

При переработке стандартного меню в ruCAD.mnu мы будем вносить много изменений.

- Устранять, по возможности, ошибки неграмотной локализации и локализовывать дополнительные пункты меню, например, относящиеся к командам утилит Express Tools.
- Удалять безжалостно все пункты меню, не относящиеся к области действия нашей САПР, например, все, что связано с 3D-рисованием и визуализацией. Пользователь, которому потребуются удаленные нами стандартные команды, в любой момент сможет вызвать стандартное меню.
- Изменять, с учетом многолетнего опыта, структуру меню и расположение некоторых стандартных команд, например:
  - добавим в контекстное меню опции для выбора объектов (почему-то отсутствующие в стандартном меню);
  - перенесем меню **Вставка** (раздел `POP4`) в меню **Файл | Импорт**;
  - меню **Правка** (раздел `POP2`) и **Вид** (раздел `POP3`) оставим на своих местах;
  - на месте раздела `POP4` вставим меню **Слои**, в которое поместим много своих полезных команд;
  - на месте раздела `POP5` вставим свое меню **Рисуй**, в котором соберем все команды рисования общего назначения;
  - шестое место в главном меню (раздел `POP6`) отведем для меню **СпецРис**, в котором, с разбивкой на тематические страницы, соберем команды специального рисования (этих команд будет очень много);
  - на седьмом месте (раздел `POP7`) поместим меню **Редакт**, в котором соберем все команды редактирования;
  - на восьмом месте (раздел `POP8`) разместим меню **Расчеты**;
  - на девятом месте (раздел `POP9`) вставим свое меню **Архив**, в котором будут сконцентрированы команды, реализующие элементы документооборота (работа с архивами проектной документации и топографическими материалами);
  - на десятом месте (раздел `POP10`) разместим меню **Настройки**, в котором соберем все команды, связанные с настройкой параметров AutoCAD и ruCAD;
  - на одиннадцатом месте (раздел `POP11`) разместим стандартное меню **Window** (Окно);
  - на двенадцатом месте (раздел `POP12`) разместим меню **Справка**, дополненное собственными командами;
  - на тринадцатом месте (раздел `POP13`) разместим меню **Профи**, в котором будут сгруппированы (с ограничением доступа) все команды, работа с которыми обычных пользователей нежелательна;
  - перенесем меню **Формат** (раздел `POP5`) в качестве подменю в меню **Сервис** — спрячем его подальше от "теток" и "обезьян с гранатами". Само меню **Сервис** упорядочим, выбросив ненужные "теткам" команды.
- Добавлять собственные команды. Команды мы будем группировать, создавая при необходимости каскадные меню. Структуру каскадных меню, как правило, будем

предусматривать с двумя-тремя уровнями вложенности, убирая с первого уровня менее популярные команды.

Как преодолеть недостатки каскадных меню

Каскадное падающее меню с вложенными подменю очень легко написать, но им очень неудобно пользоваться. Два или три уровня вложенности еще пригодны для редко используемых пунктов, но тысячи пунктов, требуемых для специализированных программ, невозможно вписать в каскадные меню, если заботиться об удобстве работы.

Для того чтобы преодолеть недостатки каскадных меню, мы поступим следующим образом:

- специализированные команды разобьем по укрупненным разделам, соответствующим разделам проекта (АС, ГП, ОВ, ВК и т. п.);
- все специальные меню мы попеременно будем отображать в разделе `POP6` главного меню;
- в каждое специальное меню, пункты с подразделами этой специальности, мы будем включать переход на общее оглавление;
- из специальных меню будут вызываться не конечные команды, а только иллюстрированные XML-меню.

Такой подход отработан в течение нескольких лет и показал высокую эффективность. Приведем фрагмент файла меню с разделом `POP6` и одним специальным подразделом (теги пунктов меню не показаны для экономии) (листинг 8.1).

Листинг 8.1. Фрагмент файла ruCAD.mnu

```
***POP6
// Оглавление специальных меню
**COMMON
[СпецРис]
[~Топография...]^C^C^P(ru-menu-special-show "TOPO")
[~Генплан...]^C^C^P(ru-menu-special-show "GP")
[~Архитектурно-строительные...]^C^C^P(ru-menu-special-show "AR")
[~Оборудование...]^C^C^P(ru-menu-special-show "OBOR")
[~Инженерные сети...]^C^C^P(ru-menu-special-show "SETI")
[~Трубопроводы...]^C^C^P(ru-menu-special-show "PIPE")
[~Водопровод и канализация...]^C^C^P(ru-menu-special-show "WATER")
[~Теплоснабжение и вентиляция...]^C^C^P(ru-menu-special-show "OV")
[~Газ...]^C^C^P(ru-menu-special-show "GAZ")
[~Электрооборудование...]^C^C^P(ru-menu-special-show "ELECTRO")
[--]
[~Спецменю моей группы]^C^C^P(ru-menu-special-show (ru-user-group))
[~Все спецрисунки!\t>>>>]^C^C^P(defun C:RU()(ru-xml-pop-mnu +
"spec_draw" "Все специальные средства")(princ));RU
**GAZ
[СпецРис-ГАЗ]
[~Плиты\t>>>>]^C^C^P(defun C:RU()(ru-xml-pop-mnu "gaz_pli" +
"Плиты газовые")(princ));RU
```

```
[~Водонагреватели\t>>>>]^C^C^P(defun C:RU()(ru-xml-pop-mnu +
"gaz_bol" "Водонагреватели")(princ));RU
[~Баллоны\t>>>>]^C^C^P(defun C:RU()(ru-xml-pop-mnu "gaz_bal" +
"Баллоны")(princ));RU
[~Резервуары\t>>>>]^C^C^P(defun C:RU()(ru-xml-pop-mnu "gaz_rez" +
"Резервуары")(princ));RU
[~Электрохимзащита\t>>>>]^C^C^P(defun C:RU()(ru-xml-pop-mnu "gaz_exz" +
"Электрохимзащита")(princ));RU
[~Трубы в 1 линию\t>>>>]^C^C^P(defun C:RU()(ru-xml-pop-mnu "TR_1l" +
"Трубы в 1 линию")(princ));RU
[~Врезаемые детали\t>>>>]^C^C^P(defun C:RU()(ru-xml-pop-mnu "gaz_vrez" +
"Врезаемые детали")(princ));RU
[~Свободные детали\t>>>>]^C^C^P(defun C:RU()(ru-xml-pop-mnu "gaz_sd" +
"Свободные детали")(princ));RU
[~Таблицы ГАЗ\t>>>>]^C^C^P(defun C:RU()(ru-xml-pop-mnu "tabl_gs" +
"Таблицы ГАЗ")(princ));RU
[~Все по газу\t>>>>]^C^C^P(defun C:RU()(ru-xml-pop-mnu "GAZ" +
"Спецрисунки газового оборудования")(princ));RU
[~Все трубопроводы\t>>>>]^C^C^P(defun C:RU()(ru-xml-pop-mnu "TRUBA" +
"Спецрисунки трубопроводов")(princ));RU
[--]
[Специальные меню...]^C^C^P(ru-menu-special-common-show)
[~Спецменю моей группы]^C^C^P(ru-menu-special-show (ru-user-group))
```

Пояснения к макросам специальных меню:

- `(ru-menu-special-show "TOPO")` — выводит в раздел `POP6` специальное меню, названное TOPO, заменяя прежнее меню;
- `(ru-menu-special-show (ru-user-group))` — выводит в раздел `POP6` специальное меню рабочей группы пользователя;
- `(ru-menu-special-common-show)` — выводит в раздел `POP6` меню COMMON (оглавление специальных меню);
- `(ru-xml-pop-mnu "ИМЯ_XML_ФАЙЛА" "Заголовок диалога")` — выводит соответствующее иллюстрированное дерево меню *(см. далее)* с командами.

Почему мы отказываемся от слайдовых меню

Во времена DOS-версий системы AutoCAD графические (слайдовые) меню (раздел `IMAGE`) были единственным средством, позволявшим проиллюстрировать, что будет нарисовано при вызове команды. Слайдовые меню являлись основными во многих прикладных системах, особенно там, где трудно описать словами результат выполнения команды. Многие разработки до сих пор построены именно так.

Но слайдовые меню имеют недостатки, связанные не столько со сложностью, сколько с трудоемкостью подготовки таких меню. Назовем эти недостатки:

- необходимо готовить множество слайдов, причем для слайдов требуется создавать специальный видовой экран с соотношением сторон 1:1.5;
- при отрисовке "фотографируемых" изображений необходимо учитывать, что закрашенные с помощью штриховки SOLID (заливки) участки не будут правильно отображены на слайде;
- из сотен слайдов необходимо собирать десятки библиотек слайдов;

- раздел IMAGE и его подразделы необходимо писать с учетом возможности перелистывания страниц слайдового меню.

Когда-то мы написали много программ для автоматизации этих работ, но это нудное занятие до того нам надоело, что мы сильно разозлились и, при переходе на работу под Windows, сразу же придумали иллюстрированное дерево меню и навсегда отказались от слайдов. В Autodesk прекрасно понимают архаичность слайдовых меню и в современных версиях AutoCAD оставляют их только для совместимости, сохранив в качестве примера только три таких меню. В диалоговом окне команды BHATCH, Центре управления и панели **Tool Palettes** (Палитры инструментов) уже используются автоматически генерируемые растровые миниатюры.

Как управлять доступом к пунктам меню

Для того чтобы сделать пункт меню недоступным, необходимо в начале текста пункта меню вставить символ ~ (тильда). Практическое значение имеет только управление доступом программным путем. Сделать это можно двумя способами: включением в текст пункта выражения на языке DIESEL и из LISP-программы.

Использование выражений языка DIESEL

В стандартном меню AutoCAD имеются примеры запрета пунктов меню, например:

```
[$(if,$(eq,$(getvar,tilemode),1),~)&Polygonal Viewport]^C^C_-vports _p
```

В этом примере пункт меню **Polygonal Viewport** недоступен, если значение системной переменной TILEMODE равно 1. Анализ значения переменной и подстановка тильды производится в выражении языка DIESEL:

```
$(if,$(eq,$(getvar,tilemode),1),~)
```

Это выражение можно интерпретировать на "человеческом" языке так: "Если системная переменная TILEMODE в момент вывода пункта меню на экран имеет значение 1, то добавить в начало текста пункта меню знак ~, иначе ничего не добавлять".

Выражение языка DIESEL в тексте пункта меню выполняется каждый раз, когда меню выводится на экран. Это очень удобно, т. к. позволяет, без всяких дополнительных усилий со стороны разработчика, управлять доступом к пунктам меню в текущий момент. К большому сожалению, язык DIESEL знает только системные переменные AutoCAD, включая USERxx, но не может анализировать значения иных переменных, даже глобальных. Все системные переменные уже несут полезную нагрузку, а пользовательских переменных USERxx мало, и дело даже не в их количестве, а в том, что разные программы (иногда даже одного автора) используют переменные USERxx для разных целей. Продуманную схему работы большой системы легко может нарушить даже одна "левая" утилита. "Вежливые" программы должны "уступать дорогу нахалам", и не вступать в разборки о том, какая программа "важнее".

В *главе 3* мы уже писали, что хорошо бы стандартизировать использование системных переменных и даже предлагали зарезервировать для стандартных целей системную переменную USERS5. В ней в виде строки длиной 250 символов могли бы храниться логические переменные доступа к различным привилегиям. Символ 0 в лю-

бой позиции строки соответствует запрещению, а любой другой символ — разрешению на привилегию с номером позиции символа.

В этом случае управление доступом к пунктам меню могло бы выполняться примерно так:

```
[$(if,$(eq,$(substr,$(getvar,USERS5),29,1),0),~)Выполнить]
```

DIESEL-выражение в этом пункте меню можно интерпретировать так: "Если 29-й символ строковой системной переменной USERS5 в момент вывода пункта меню на экран имеет значение 1, добавить в начало текста пункта меню знак ~, иначе ничего не добавлять". Значение переменной USERS5 могло бы формироваться программным путем в зависимости от различных факторов.

В своих прежних разработках мы также грешили использованием USERxx, в системе ruCAD мы постараемся от вредных привычек избавиться.

Использование языка LISP для управления доступом

Из LISP-программ также можно управлять доступом к пунктам меню. Для этого используется функция AutoLISP `menucmd`, о некоторых возможностях которой мы уже писали.

Возможны два варианта управления доступом:

- по абсолютным ссылкам на пункт меню с использованием номеров падающих меню и номеров пунктов;
- по относительным ссылкам на пункт меню с использованием его тега.

По первому варианту выражение

```
(menucmd "P1.2=~")
```

запрещает доступ ко второму пункту падающего меню `POP1` (что бы там ни находилось), а выражение

```
(menucmd "P1.2=")
```

разрешает доступ ко второму пункту падающего меню `POP1` (вернее, снимает с него все дополнительные пометки).

Абсолютный доступ, конечно, не очень удобен, т. к. нет никакой гарантии, что в пункте меню с таким порядковым номером окажется ожидаемое разработчиком содержание. Например, в AutoCAD R12 таким способом были защищены некоторые пункты стандартного меню и при его корректировке без видимых причин становились недоступными попавшие на эти места другие команды.

По второму варианту выражение

```
(menucmd "Gacad.ID_help=~")
```

запрещает доступ к пункту падающего меню с тегом `ID_help` группы `acad` (где бы этот пункт ни находился), а выражение

```
(menucmd "Gacad.ID_help=")
```

разрешает доступ к пункту падающего меню с тегом `ID_help` группы `acad` (вернее, снимает с него все дополнительные пометки).

Этот вариант гораздо лучше и именно его мы будем использовать. Естественно, все пункты меню, которыми мы желаем управлять, должны иметь обязательный тег. Значение тега используется и для вывода соответствующей подсказки из раздела `HELPSTRINGS`.

При программном управлении доступом необходимо учитывать, что:

- в тексте пункта меню не должно быть никакого DIESEL-выражения;
- при нескольких одинаковых тегах (а такое может быть для однотипных пунктов меню с одинаковой строкой подсказки) функция `menucmd` действует только на первый из пунктов с одинаковыми тегами.

Это плохо, т. к. хотелось бы за один раз устанавливать режимы для многих пунктов меню.

Но "если нельзя, но очень хочется, то можно". Можно потому, что кроме средств AutoLISP мы имеем доступ ко всем объектам системы AutoCAD через механизмы ActiveX. Пункты меню (объекты `PopupMenuItem`) имеют свойства `Enabled` и `Check`, а если такие свойства есть, то мы ими можем управлять. Функции для управления меню мы рассмотрим в *главе 13*.

Как ставить "галочки" в меню

Некоторые пункты меню нужно выбирать или иначе помечать "галочками". Это могут быть пункты, показывающие текущее состояние переключателя, срабатывающего при выборе команды, или просто напоминания о каком-то свойстве. Например, в оглавлении специальных меню мы будем помечать пункт специального меню рабочей группы, в которую входит пользователь. Пользователи могут по своему желанию изменять рабочую группу и такое напоминание, как показывает практика, не будет излишним.

Простановка отметок выполняется точно так же, как и управление доступом к меню, только вместо знака тильды ставится восклицательный знак. Функции для простановки отметок мы также рассмотрим в *главе 13*.

Как правильно использовать экранное меню

В ранних версиях AutoCAD экранное (боковое) меню всегда было на экране. Разработчики интенсивно использовали его в своих программах. Большинству пользователей всегда удобней щелкнуть по пункту экранного меню, чем набирать на клавиатуре опции команд. За отображаемой частью меню часто прятались какие-то данные, например:

```
**WALL 3
[Толщина]
[стены мм]
[ 1 кирпич ]^P250.0;
[1.5 кирпича ]^P380.0
[2 кирпича ]^P510.0
[2.5 кирпича ]^P640.0
```

В приведенном примере в экранном меню отображается толщина стены в кирпичах, а возвращается число (толщина в миллиметрах). Мог быть и такой вариант:

```
[2 кирпича ]^P510.0;0.93
```

В этом случае возвращалось бы два числа (символ ";" в макроопределениях меню эквивалентен нажатию клавиши <Enter>).

Такое меню могло использоваться в следующем фрагменте программы:

```
(menucmd "S=WALL")
(setq width (getreal "\nВыбери толщину стены из экранного меню: "))
(setq Rh (getreal))
```

В этой программе пользователь *обязан* был выбрать данные именно из меню, которое *обязано* было появиться на экране. При этом вводилась не только толщина стены в ответ на видимый запрос, но и, тайком от пользователя, второе число (коэффициент теплопередачи).

Иногда на выборе данных из экранного меню была построена вся идея программы.

В современных версиях системы AutoCAD пользователь может спрятать экранное меню ради увеличения рабочего поля экрана. Рассчитывать на то, что оно всегда будет видимым (или требовать этого от пользователей) нельзя. К счастью, в AutoCAD теперь появились "умные" контекстные меню. При щелчке правой кнопкой мыши над зоной графического экрана во время выполнения команд появляется контекстное меню, в котором, помимо стандартных пунктов (обычно Enter, Cancel, Pan, Zoom), имеются опции соответствующей команды. Следовательно, экранное меню можно безболезненно отключить.

Но как же быть при разработке своих программ? Писать свои контекстные меню и вызывать их в нужный момент? Можно и так, только как этот нужный момент определить? Принудительно показывать выводить меню можно выражением

```
(menucmd "P0=POPxxx")(menucmd "P0=*"),
```

которое *покажет* контекстное меню `POPxxx`. Но пользователю это может не понравиться. Он должен иметь возможность сам управлять своими действиями.

Простое решение, о котором не все знают, заключается в следующем. При формировании командной строки необходимо ее опции, перечисляемые через "/", заключать в *квадратные скобки*. В результате в контекстное меню будут автоматически включаться именно эти пункты (листинг 8.2). Вот как все просто! Жидкие аплодисменты программистам Autodesk! Аплодисменты могли бы быть бурными и даже переходящими в овации, если бы контекстные меню всегда правильно обрабатывали русскоязычные опции.

Замечание

В контекстном меню отображается содержимое опций в квадратных скобках, а не то, что записано в функции `initget`, вызов которой устанавливает допустимые опции и обычно предваряет интерактивный ввод данных.

Попытка вызова несуществующего контекстного меню воспринимается системой AutoCAD как нажатие клавиши <Enter>.

Листинг 8.2. Пример автоматического создания опций контекстного меню

```
(defun test-popup-menu ( / kw do_next)
(setq do_next T)
(while do_next
(initget "Первый Второй Третий")
(setq kw (getkword "\nОпции без скобок - Первый/Второй/Третий: "))
(print kw)
(initget "Первый Второй Третий Хватит")
;;; Так надо записывать опции
(setq kw (getkword "\nСо скобками [Первый/Второй/Третий/Хватит]: "))
(print kw)
(initget "1-й 2-й Третий Выход")
;;; Неправильные опции для контекстного меню
;;; (отличаются от заданных в initget)
(setq kw (getkword "\nВыбери [1-й/2-й вариант/Третий вариант/Выход]: "))
(print kw)
(if (= kw "Выход")(setq do_next nil))
)
)
```

Как мы будем писать пункты меню ruCAD

Пункт меню имеет следующую структуру:

```
<тег> [<текст пункта>]<макрос>
```

Здесь `<тег>` — это необязательный набор буквенно-цифровых символов и символа подчеркивания (_), предназначенный для идентификации пункта меню. Теги могут повторяться, хотя это и нежелательно. Теги используются для вывода соответствующей строки подсказки и для управления доступом к пунктам меню.

`<текст пункта>` — это текст, отображаемый при выводе меню на экран. В тексте пункта могут быть ссылки на подменю, отмечаемые символами "—>" в начале наименования этого подменю, и окончания подменю — символами "<—" в начале наименования последнего пункта, входящего в подменю. Пункты, имеющие в качестве наименования два минуса, — это разделительные строки, которые в падающем меню отображаются в виде горизонтальных разделительных линий.

`<макрос>` — строка, описывающая операции, которые выполняет пункт меню (макрос). В макрос могут входить команды системы AutoCAD и их опции, LISP-выражения, а также специальные комбинации символов, имитирующих, например, нажатие клавиш. Длинные макросы могут занимать более одной строки. В этом случае в конце всех строк, кроме последней строки макроса, ставится знак плюс как символ продолжения. Длина текста макроса не должна превышать 255 символов.

Совет

Не используйте в макросах пробелы, как это сделано в стандартном меню. Пробел (а количество пробелов трудно визуально контролировать) эмулирует нажатие клавиши <Enter>, а это можно сделать с помощью символа ";".

Если потребуется в меню записать путь к файлу (со слэшами), то не используйте ни двойной слэш, ни обратный, даже если они находятся внутри строки, передаваемой

в LISP-функцию. Прежде чем вступят в действие "законы LISP", текст макроса будет обрабатываться по "законам меню", в которых слэш считается управляющим символом (означает паузу для ввода). Заменяйте слэш выражением `(chr 92)`. Например, так:

```
[Загрузить]^C^C(load (strcat "menu" (chr 92) "my.lsp"))
```

Мы будем формировать пункты меню по собственным правилам.

Правила формирования тегов пунктов меню ruCAD

Все теги "наших" пунктов меню мы будем начинать с префикса RU_. Для пунктов меню, доступ к которым ограничен в соответствии с системой прав пользователей, применяются специальные префиксы (табл. 8.1).

Таблица 8.1. *Специальные префиксы тегов*

Префикс	Наименования прав и привилегий
RU_ADMIN	Администрирование
RU_DEVELOP	Разработка приложений
RU_XML_EDIT	Редактирование XML
RU_PROFI	Права профессионала
RU_GET_PROJ	Брать типовой проект из библиотеки
RU_PUT_PROJ	Помещать типовой проект в библиотеку
RU_SETUP_CAD	Изменение настроек ruCAD
RU_SETUP_LAY	Изменение настроек слоя
RU_GET_LAY	Брать кальки из архива
RU_PUT_LAY	Сдавать кальки в архив
RU_GET_TOPO	Брать топографию из архива
RU_PUT_TOPO	Сдавать топографию в архив

Такая система позволит программно управлять доступом к пунктам меню.

Теги пунктов меню стандартных команд AutoCAD мы будем оставлять без изменений, за исключением команд, доступных только в AutoCAD 2004. В теги этих пунктов меню мы будем включать символы "2004". Наличие этих символов позволит программно запрещать доступ к этим командам, если работа производится с системой AutoCAD 2002.

Правила формирования текстов пунктов меню ruCAD

Тексты всех пунктов меню мы будем писать на русском языке, включая стандартные команды AutoCAD. Для стандартных команд, помимо наименования пункта, мы будем включать имя команды на английском языке, выравнивая его по правому краю с помощью управляющих символов `\t`. Включать будем именно имя команды, а не текст пункта на английском языке. Например, если в англоязычном меню име-

ется пункт **Multiline**, которому соответствует команда MLINE (МЛИНИЯ), то в меню ruCAD текст аналогичного пункта будет написан так:

```
[Мультилиния\tMline]
```

Наличие оригинальных имен команд в тексте пункта позволит визуально выделить стандартные команды, сделает меню более информативным и даст возможность пользователям постепенно запоминать их настоящие названия.

Команды библиотеки Express Tools, которые мы будем включать в свое меню (причем не одной группой, а в соответствующие разделы), также будем сопровождать именем команды и ссылкой на библиотеку Express, например:

```
[~Выровнять пространство\tAlignSpace (Express)]
```

Все пункты меню с командами Express Tools мы будем включать с запретом доступа (знак тильды в начале текста пункта). Эта библиотека, хотя и доступна для бесплатного использования (а в AutoCAD 2004 входит в инсталляционный комплект), но может быть и не установлена на компьютер пользователя. При инициализации системы ruCAD, если библиотека Express Tools будет обнаружена, то доступ к ее пунктам меню будет программно разблокирован.

Все пункты "наших" команд также будут изначально заблокированы, а разблокировка доступа будет производиться при старте системы. Это сделано для того, чтобы пользователь, загрузивший меню ruCAD в обычную систему AutoCAD сразу же видел, что забрел не по адресу.

Так как очень много пунктов нашего меню будут загружать иллюстрированное дерево меню, мы установим для них специальное обозначение ">>>>" и подобные пункты меню будут выглядеть так:

```
[~Водонагреватели\t>>>>]
```

Правила формирования макросов пунктов меню ruCAD

Макросы стандартных команд AutoCAD и Express Tools мы будем оставлять без изменений. Но, для повышения динамичности работы, в макросах некоторых стандартных команд мы предусмотрим автоматическое повторение команды. Это достигается вставкой символа `*` в начале макроса, например:

```
[*&Отрезок\tLine]*^C^C_line
[*По&добие\tOffset]*^C^C_offset
```

Звездочку в тексте пункта мы поставили просто как метку, чтобы обратить внимание на необычное поведение команды. В AutoCAD R10 в падающем меню большинство команд было с автоматическим повторением, а в экранном меню — без повторения. В современных версиях фирма Autodesk отказалась от такой практики, хотя возможность повторения команд осталась.

Примечание

Если значение системной переменной MENUECHO равно 2, то первое приглашение повторяемых команд на экран не выводится.

О сочетании MENUECHO и ^P

Системная переменная MENUECHO содержит код, являющийся суммой установленных битовых флагов управления видимостью эхо-вывода (печати в командной строке) текстов макросов меню. Ее начальное значение: 0 (все битовые флаги сброшены). Используются следующие битовые флаги:

- 1 — подавление эхо-вывода пунктов меню (аналогично первому вхождению переключателя `^P` в макрос пункта меню);
- 2 — подавление системных запросов во время выполнения команд из меню;
- 4 — подавление переключателя эхо-вывода пунктов меню (аналогично второму вхождению переключателя `^P` в макрос);
- 8 — отображение всех выводимых и вводимых строк (режим отладки для DIESEL-выражений).

Устанавливать значение переменной MENUECHO мы будем в специальной функции инициализации системы и это значение нужно выбрать оптимальным. На отображение эхо-вывода макросов влияет и специальная комбинация символов `^P`, которая может включаться в текст макроса. Нам желательно, чтобы тексты макросов в командной строке не отображались в обычном режиме, но могли бы отображаться при включении отладочного режима. В макросах стандартных пунктов меню в последних версиях переключатель `^P`, как правило, отсутствует (за исключением макросов с LISP-выражениями). Сделано это, вероятно, для того, чтобы пользователи постепенно запоминали имена команд. В пунктах меню с переключателем `^P`, он вписывается два раза для отключения эхо-вывода и восстановления предыдущего режима, например:

```
[На передний план]^C^C^P(ai_draworder "_f") ^P
```

Для того чтобы выбрать самый удобный вариант, мы написали несколько тестовых пунктов меню, выполнили их при различных значениях системной переменной MENUECHO, проанализировали результаты и свели их в таблицу (табл. 8.2).

Таблица 8.2. *Режимы эхо-вывода макросов меню*

Макрос	Отображение макроса при значении системной переменной MENUECHO					
	0	**1**	**2**	**4**	**8**	**9**
`[Тест 1]^C^C(prompt "Тест 1")`	Да	Нет	Да	Да	Да	Нет
`[Тест 2]^C^C^P(prompt "Тест 2")`	**Нет**	**Да**	Нет	Да	Нет	Да
`[Тест 3]^C^C^P(prompt "Тест 3");^P`	**Нет**	**Да**	Нет	Да	Нет	Да
`[Отрезок]^C^C_line`	Да	Нет	Да	Да	Да	Нет

Для обеспечения возможности использования команд с автоматическим повторением (с символом `*` в начале макроса) мы проверили и режим вывода сообщений таких команд (табл. 8.3).

Таблица 8.3. *Режимы вывода сообщений команд при автоповторении*

Макрос пункта меню с автоматическим повторением команды	Вывод сообщений команд при значении системной переменной MENUECHO					
	0	1	2	4	8	9
[*Отрезок]*^C^C_line	Да	Да	Нет	Да	Да	Да

На основании анализа результатов экспериментов мы можем сделать следующие выводы:

- значение переменной MENUECHO=2 отвергаем из-за потери первых сообщений команд, повторяемых автоматически;
- оптимальным вариантом является установка MENUECHO=0 и вставка переключателя `^P` только в начале макроса.

Вставка переключателя `^P` в начало и конец макроса усложняет меню и требует вставки дополнительного эмулятора клавиши <Enter>. Такой вариант является потенциальным источником ошибок и мы от него отказываемся.

Выбранный режим дает следующие преимущества:

- стандартные пункты меню работают в стандартном режиме;
- тексты наших макросов не отображаются в командной строке в обычном режиме работы и будут отображаться при переходе в режим отладки с установкой MENUECHO=1;
- обеспечивается возможность использования автоматического повторения команд.

Синтаксис наших макросов

Собственные макросы мы будем писать особым образом. Любые нестандартные команды представляют собой вызов ранее определенной функции или загрузку требуемого файла с последующим вызовом необходимой функции. Обычно нестандартные макросы пунктов меню пишут так:

```
[Моя команда]^C^CМОЯ_КОМАНДА
```

если предполагается, что функция `C:МОЯ_КОМАНДА` уже загружена (так сделано, например, в библиотеке Express Tools), или так:

```
[Моя команда]^C^C(if(null C:МОЯ_КОМАНДА)(load "my_command"));МОЯ_КОМАНДА
```

И первый, и второй варианты, конечно, могут быть более сложными.

Наши макросы будут строиться по следующим схемам:

```
[~Схема 1]^C^C^P(имя_функции [параметры])
```

По этой схеме производится обычный вызов, возможно с параметрами, функций из резидентной библиотеки. Загрузка этой функции гарантирована всегда, иначе пункт меню остается недоступным.

```
[~Схема 2]^C^C^P(defun C:RU()(ru-app-load "file_name"));RU
```

По схеме 2 загружается файл file_name (расширение не указывается), в котором имеется вызов функции, которая должна выполняться сразу после загрузки файла.

Обычно это вызов функции `start` без параметров. Загрузка файла обернута в функцию-команду `C:RU`, которая вызывается сразу же после определения. Таким образом, после каждого щелчка по пункту меню мы имеем новое определение команды "RU".

```
[~Схема 3]^C^C^P(defun C:RU()(if(ru-app-load "file_name")+(start params)));RU
```

По схеме 3 макрос работает так же, как и по схеме 2, только функция `start` вызывается с параметрами. Проверка загрузки производится для того, чтобы в случае сбоя не вызывалась предыдущая не переопределенная функция `start`.

Почему мы делаем единственную команду, переопределяемую при каждом обращении к меню? Почти всегда, и у Autodesk, и у других разработчиков в загружаемом файле определены одна или несколько функций с префиксом `C:`, которые потом можно использовать так же, как и стандартные команды, а именно:

- вызывать команду по имени, набрав имя на клавиатуре;
- повторять последнюю выполненную команду, нажав клавишу <Enter>.

На этом приятные особенности функции `C:МОЯ_КОМАНДА` исчерпываются.

Рассмотрим недостатки таких команд:

- в функцию `C:МОЯ_КОМАНДА` нельзя передавать параметры;
- таких имен будет очень много, в реальных системах эти имена даже придумать сложно, тем более невозможно запомнить.

Как, например, придумать имя команде, которая выполняет "Рисование вентиля фланцевого на вертикальном трубопроводе в боковой плоскости аксонометрической схемы"? ВЕНТФЛВЕРТБОКИЗ или ВФЛВБИЗ? А подобных ситуаций у нас будут тысячи. Пользователю было бы даже легче запомнить цифровые номера команд, чем маловразумительные сокращения. Заставлять пользователя запоминать что-то подобное бессмысленно. Значит, этого мы и не будем делать. Вместо тысяч "плохих" или "хороших" имен команд мы введем одно-единственное имя команды — RU. Простенько и со вкусом. А если и это имя нам не понравится, мы его всегда сможем заменить, ведь оно единственное. Пользователю вообще никогда не нужно набирать имя команды на клавиатуре. Единственным исключением является возврат в меню ruCAD после временной загрузки стандартного меню, да и то если мы поленимся (или побоимся) вставить в стандартное меню пункт для возврата.

Вызов команд всегда будет производиться через меню.

А зачем мы каждый раз заново загружаем файл, это же задержка работы? Задержка эта мизерная, пользователь ее даже не заметит, но он всегда будет работать с самой свежей версией программы. Первыми пользователями на этапе разработки будем мы сами. Мы будем десятки раз изменять исходные тексты (если вы про себя думаете иначе, то вы себе льстите), и каждый раз нам не потребуется дополнительная перегрузка. Когда мы будем рассматривать функцию `ru-app-load`, вы увидите, как изощренно она написана. Эта функция, в режиме разработки, даже будет перекомпилировать программу из исходных текстов.

Итак, что мы имеем положительного от использования схемы 1 или схемы 2:

- каждый пункт меню является псевдокомандой, которую можно повторить нажатием клавиши <Enter>;
- в команде, определяемой в макросе, мы можем предусматривать вызов функций с параметрами.

Как создавать панели инструментов

Только не так, как описано в документации и в книгах по AutoCAD! Использование диалогового окна **Customize** (Адаптация) мы оставим обычным пользователям, а сами будем редактировать описание панелей инструментов с помощью простого текстового редактора в файле ruCAD.mnu. Описания панелей инструментов сосредоточены в разделе TOOLBARS.

Прежде всего, мы выбросим из него все ненужные нам стандартные панели инструментов. Из оставшихся стандартных панелей мы выбросим кнопки с редко используемыми в двухмерном строительном проектировании командами. После этого мы дополним панели своими кнопками. Теоретически необходимо для каждой команды создать свою кнопку на соответствующей панели, но команд у нас будет тысячи, панелей инструментов потребуется сотни. А где же пользователи будут рисовать, если весь экран способны закрыть десяток панелей? Сделаем кнопки только для самых популярных команд.

Мы сделаем четыре основных панели, сознательно увеличив их длину (максимум до 30 кнопок). В тексте меню координаты панелей мы запишем такие, чтобы в исходном состоянии панели находились в верхней части экрана. При установленном разрешении экрана 800×600 пикселов. Вообще все эксперименты с размещением кнопок и панелей мы будем проводить при установленном разрешении экрана 800×600 пикселов (по крайней мере, проверять результаты при таком разрешении).

Основные панели инструментов меню ruCAD:

- TB_STANDARD_RU;
- TB_DRAW_RU;
- TB_EDIT_RU;
- TB_OBJECT_PROPERTIES_RU.

Для прочих панелей желательно на каждой панели располагать не более 13 кнопок — в этом случае панель можно расположить вертикально у бокового края экрана.

В четырех основных панелях мы сосредоточим самые популярные (для круга наших пользователей) команды. Какие именно — вопрос сложный. В течение нескольких лет мы отслеживали частоту употребления команд и сделали определенные выводы. У наших читателей популярными могут оказаться совсем другие команды. Приводить текст файла меню сейчас мы не будем (вы найдете его на компакт-диске), т. к. еще при написании системы степень популярности команд может измениться.

Синтаксис описаний панелей инструментов

Описание панели инструментов в общем виде имеет следующую структуру:

```
**МЕТКА
тег_панели [_Toolbar("Заголовок панели",ориентация,видимость,X,Y,ряды)]
тег_кнопки [_Button("Подсказка для кнопки",пиктограмма_16, пиктограмма_32)]макрос
тег_групповой_кнопки [_Flyout ("Подсказка для кнопки", пиктограмма_16,
пиктограмма_32, видимая_пиктограмма,псевдоним_выпадающей_панели)]макрос
[--]
тег_списка [_Control (тип_списка)]
```

Выше мы привели схему описания панели с четырьмя возможными элементами — обычной кнопкой (Button), групповой кнопкой (Flyout), разделителем (--) и раскрывающимся списком стандартных элементов.

**МЕТКА — строка-идентификатор панели инструментов (аналог подраздела падающих меню). После загрузки MNU-файла создается MNS-файл и в него добавляется вторая строка идентификатора, в которой с префиксом TB_ записан, с переводом в верхний регистр и заменой пробелов на символ подчеркивания, заголовок панели, извлеченный из строки определения свойств панели. Например, если в MNU-файле было записано

```
**TB_DRAW_RU
```

то в MNS-файле будет записано

```
**TB_DRAW_RU
**TB_РИСОВАНИЕ_RUCAD
```

если заголовок панели был определен как "Рисование ruCAD".

Тег (любой) — строка-идентификатор, имеющая такое же назначение, как и у пунктов меню *(см. ранее)*.

Макрос — строка, описывающая операции, которые выполняются при щелчке по кнопке.

[--] — декоративный разделитель (линия) между группами кнопок в панели.

Тип_списка — одно из ключевых слов: _Color, _Dimstyle, _Layer, _Linetype, _Lineweight, _PlotStyle, _Refblkname, _UCSManager, _View и _ViewportScale, определяющих тип выводимого стандартного списка (в AutoCAD 2004 к перечисленным ключевым словам добавились _txtstyle, _Undo и _Redo). К списку нельзя присоединить свой макрос. Собственный раскрывающийся список вставить в панель тоже нельзя.

Заголовок панели — это строка, включающая буквы, цифры, символы "-" и "_". Выводится в качестве заголовка окна панели, когда она находится в плавающем положении, а также в списках панелей инструментов в контекстном меню и в диалоговом окне **Customize** на вкладке **Tooolbars**. Может использоваться в качестве псевдонима для программного доступа.

Ориентация — одно из следующих ключевых слов: _Floating (панель находится в плавающем положении), _Top (панель зафиксирована в верхнем горизонтальном положении), _Bottom (панель зафиксирована в нижнем горизонтальном положении), _Left (панель зафиксирована в левом вертикальном положении) или _Right (панель зафиксирована в правом вертикальном положении). Определяет расположение панели на экране.

Видимость — одно из ключевых слов: _Show (панель видимая) или _Hide (панель не видна на экране). В меню ruCAD видимыми мы сделаем только четыре основных панели.

X — число (в пикселах), указывающее положение левой кромки панели относительно левого края экрана.

Y — число (в пикселах), указывающее положение верхней кромки панели относительно верхнего края экрана.

`Ряды` — количество рядов кнопок при нахождении панели в плавающем положении.

`Подсказка для кнопки` — строка всплывающей подсказки для кнопки.

`Пиктограмма_16` и `пиктограмма_32` — имена пиктограмм в малом и большом форматах *(подробнее см. далее)*.

`Видимая_пиктограмма` — одно из ключевых слов: `_OwnIcon` (собственная пиктограмма групповой кнопки) или `_OtherIcon` (пиктограмма последней нажатой кнопки). Определяет, какая пиктограмма отображается на групповой кнопке *(о групповых кнопках см. далее)*.

`Псевдоним_выпадающей_панели` — ссылка на метку панели инструментов, входящую в данную группу. Например, описание

```
ID_TbOsnap [_Flyout("Объектная привязка", ICON_16_OSNAP, ICON_16_OSNAP, _OtherIcon,
RUCAD.TB_OBJECT_SNAP_RU)]
```

привязывает к кнопке панель TB_OBJECT_SNAP_RU из группы RUCAD и выводит (с черным треугольником) последнюю нажатую кнопку.

Работа с *групповыми кнопками* довольно сложна. Выбирать из разворачивающейся панели кнопок следует скольжением по развернувшимся пиктограммам с нажатой кнопкой мыши. Даже описать это непросто, тем более трудно выполнить, особенно "тетке", которая держит мышь двумя руками. Мы постараемся свести количество таких кнопок до минимума. Например, в стандартной панели мы вытащим из раскрывающегося ряда кнопок "Зумирование" кнопки для операций ZOOM P (Покажи Предыдущий) и ZOOM E (Покажи Границы), применяемых очень часто.

Как сохраняется конфигурация панелей инструментов

Если пользователь перетащит панель инструментов на другое место, то новое положение панели запоминается в реестре, например:

```
[HKEY_CURRENT_USER\SOFTWARE\Autodesk\AutoCAD\R%Номер_Версии%
\ACAD-%Серийный_Номер%\Profiles\ruCAD\Toolbars]
"ruCAD.TB_STANDARD_RU"="show float 19 214 2"
```

Если пользователь воспользуется командой CUSTOMIZE, то новое положение панелей и кнопок будет записано в MNS-файл. Для того чтобы сохранить удачную конфигурацию панелей навсегда, необходимо скопировать новое определение панелей из MNS-файла в MNU-файл. Кроме того, может потребоваться удаление указанного выше раздела реестра.

Как хранить значки для панелей инструментов

Значки (пиктограммы) для панелей инструментов можно хранить двумя способами:

- в виде отдельных BMP-файлов, размещенных в каталоге на путях поиска AutoCAD;
- в DLL-библиотеке, размещенной рядом с файлом меню и имеющей такое же имя.

В первом случае описание кнопки на панели инструментов в файле меню выполняется по схеме

```
[_Button("Подсказка", "FILE16.BMP", "FILE32.BMP")]МАКРОС
```

а во втором случае по схеме

```
[_Button("Подсказка", BITMAP16, BITMAP32)]МАКРОС
```

В схемах FILE16.BMP и FILE32.BMP соответствуют именам файлов малого и большого значка, а BITMAP16 и BITMAP32 — именам ресурсов типа BITMAP в DLL-библиотеке.

Если при загрузке меню соответствующее изображение не найдено, то пиктограмма заменяется улыбающейся мордочкой неизвестного существа.

Стандартные пиктограммы меню AutoCAD 2002 скомпонованы в библиотеку acadbtn.dll и имеют имена ресурсов, начинающиеся с префикса `ICON_`, а меню AutoCAD 2004 скомпонованы в библиотеку acadbtn.dll и имеют имена ресурсов, начинающиеся с префикса `RCDATA_`. В этих ресурсах уже не простые BITMAP, а пиктограммы в формате RAW.

Замечание

Формат RAW не поддерживает ничего. Не хранятся даже данные о количестве каналов, глубине цвета и разрешении. Изображение хранится просто как поток пикселов с фиксированным заголовком, куда можно впоследствии поместить любую текстовую информацию. Обычно в файлах такого стандарта сохраняется информация, которая непосредственно передается с матриц цифровых фотокамер и выводится на печать в некоторых моделях принтеров. Размеры файлов в формате RAW даже больше, чем в BMP (размер acadbtn.dll в AutoCAD 2004 — 1,98 Мбайт, в AutoCAD 2002 — 0,33 Мбайт практически с тем же набором картинок), прочитать его нельзя почти никаким просмотрщиком или редактором. Зачем фирма Autodesk применила такой формат — непонятно, возможно с прицелом на будущую многоплатформенность. Или просто картинок жалко.

Использование изображений из отдельных файлов позволяет одну и ту же картинку использовать в меню для версий 2002 и 2004. Для этого все BMP-файлы можно было бы поместить в каталог %ruCadLocalAppDataAcad% при размещении самих файлов меню в каталогах %ruCadLocalAppDataAcad%\15 и %ruCadLocalAppDataAcad%\16.

При компоновке пиктограмм в одну библиотеку мы избавляемся от всех проблем, связанных с наличием большого количества маленьких файлов, так же, как и при использовании библиотек блоков.

Как изготовить DLL-файл с пиктограммами

Сделать это можно в любой среде программирования. Например, в Visual Basic это делается так:

1. Создают новый проект ActiveX DLL.
2. Открывают Resource Editor (если редактор ресурсов не загружен, то загрузить его с помощью пункта **Add-In Manager** меню **Add-Ins**).
3. Загружают в файле ресурса каждый предварительно заготовленный BMP-файл (возможен выбор нескольких файлов), и переименовывают каждый из создаваемых ресурсов так, как он описан в меню, изменяя свойство ID.
4. Сохраняют проект с именем, соответствующим имени файла меню (в нашем случае ruCAD).
5. Выполняют пункт **Make ruCAD.dll** из меню **File**.

После этого переносят созданную DLL-библиотеку в каталог меню.

Очевидный недостаток такого способа в том, что очень неудобно переименовывать в диалоговом окне практически вслепую сотни ресурсов.

Используя Borland Delphi, можно сделать это гораздо быстрее. Собрав все BMP-файлы в один каталог, мы выполняем команду `dir *.bmp > ruCAD.rc`. Полученный список путем несложных манипуляций с *приличным* текстовым редактором (например, Ultraedit) превращаем в файл сценария ресурсов, в котором каждая строка имеет вид:

`ИМЯ_РЕСУРСА BITMAP ИМЯ_ФАЙЛА.BMP` (листинг 8.3).

Листинг 8.3. Фрагмент файла ruCAD.rc

```
ICON_16_ZOOSCA          BITMAP          ICON_16_ZOOSCA.BMP
ICON_16_ZOOVMA          BITMAP          ICON_16_ZOOVMA.BMP
ICON_16_ZOOWIN          BITMAP          ICON_16_ZOOWIN.BMP
IDB_TBAR_SMILEY BITMAP                  IDB_TBAR_SMILEY.BMP
RU_CURSOR_0_16          BITMAP          SA_00_16.BMP
RU_CURSOR_0_32          BITMAP          SA_00_32.BMP
RU_CURSOR_45_16 BITMAP                  SA_45_16.BMP
RU_CURSOR_45_32 BITMAP                  SA_45_32.BMP
```

Далее запускаем компилятор ресурсов, входящий в состав Delphi:

```
BRCC32.EXE ruCAD.rc
```

и, если сценарий написан правильно, получаем файл ruCAD.res.

Пишем вручную файл проекта для Delphi ruCAD.dpr (листинг 8.4).

Листинг 8.4. Проект ruCAD.dpr

```
library ruCAD;
{$R *.res}
begin
end.
```

Загружаем этот проект в Delphi и компилируем. В результате получили желаемую библиотеку ruCAD.dll.

Разумеется, так же можно создать DLL-файл в среде Visual C++ и любых других. Конечно, можно сделать все и сложнее, т. е. создавая файл ресурсов внутри IDE (Integrated Development Environment, Интегрированная среда разработки), добавляя и переименовывая ресурсы в диалоговых окнах.

А где брать пиктограммы?

Конечно, теоретически надо рисовать их самим. По две картинки на каждую кнопку — размером 16×15 и 32×30 пикселов. Разумеется, использовать примитивный Button Editor из штатных средств AutoCAD не стоит. Имеется много графических редакторов, позволяющих выполнять такие работы более комфортно. Например, очень удобна программа Icon Edit Pro, которая пригодится нам и для создания

значков приложений[1]. Эта программа имеет множество инструментов и позволяет загружать изображения из файлов различных форматов, в том числе извлекая картинки из файлов с расширениями exe, dll и icl.

Однако, даже имея превосходный инструмент, пиктограмму еще нужно нарисовать, а для этого нужны определенные способности и вкус. Образцы или заготовки для своих пиктограмм можно извлечь из других файлов с помощью различных редакторов ресурсов. Например, с помощью редактора ресурсов Restorator[2] можно одним щелчком мыши извлечь все bitmap-изображения из acadbtn.dll системы AutoCAD 2002, отредактировать, как надо, а затем скомпоновать из них свою библиотеку. Мы использовали пиктограммы из AutoCAD 2002 в ruCAD.dll для того, чтобы в *своем* меню заменить "мутные" пиктограммы AutoCAD 2004 для стандартных панелей инструментов.

Замечание

Редактор ресурсов Restorator позволяет очень многое — отредактировать строковые сообщения, диалоговые окна, заменить ресурсы в EXE- и DLL-файле. Именно с помощью этой программы чаще всего производится неофициальная русификация программ.

Как организовать ввод данных

Ввод данных путем выбора из меню прежде применялся очень широко, мы об этом уже упоминали в *разд. "Как правильно использовать экранное меню" настоящей главы*. От экранного меню мы решили отказаться совсем, но возможен еще ввод данных через появляющееся в нужный момент падающее меню. Реализовать это очень просто и такие идеи постоянно появляются у разработчиков. Примером крайнего выражения такого подхода являлась система ARKO, в которой на самом видном месте постоянно были доступны меню **База1** и **База2**, за пунктами которых скрывались конкретные данные. Постоянное присутствие на экране таких меню дезориентировало пользователей.

В своем меню мы не будем вообще хранить данные. При необходимости выбора из условно постоянных опций данные будут храниться в XML-деревьях с возможностью визуального выбора и корректировки. Достаточно большие и изменяемые данные мы будем хранить во внешних базах данных и обращаться к ним специальными средствами.

Как создать иллюстрированное дерево меню

Уже несколько раз мы упоминали про загадочное иллюстрированное дерево меню. Пришла пора познакомиться с ним поближе.

Иллюстрированное дерево меню (рис. 8.1) позволяет устранить многие недостатки традиционного интерфейса AutoCAD. Мы видим пример выбора программы для рисования одного из тысяч изображений. Пользователь, работая с иерархическим "деревом" меню, видит название раздела, название программы, иллюстрацию к ней

[1] **http://www.iconedit.com**.

[2] **http://www.bome.com/Restorator**.

(иллюстрацию можно увеличить) и дополнительный комментарий. При выборе пункта меню запускается соответствующая LISP-программа или выполняется стандартная команда AutoCAD. При последующем обращении *этого* пользователя к *этому* меню оно раскроется на том же месте.

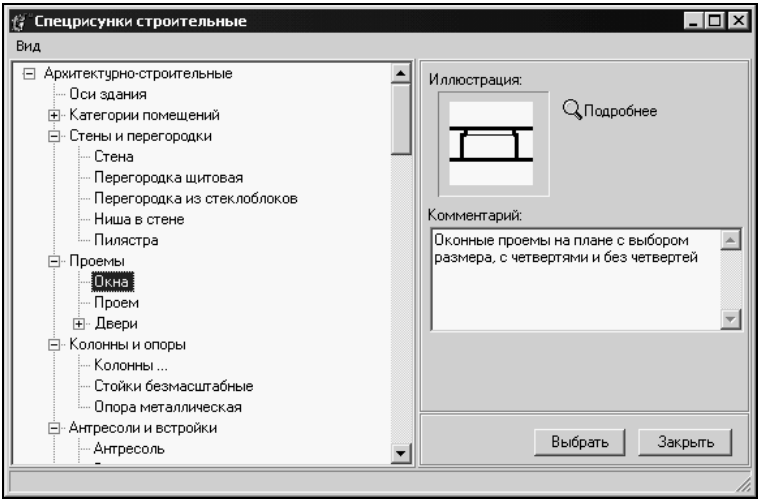


Рис. 8.1. Иллюстрированное дерево меню

Пользователь, имеющий право редактирования меню, может мгновенно изменить иллюстрацию, вставив ее из буфера обмена или прямо вырезав с экрана AutoCAD, изменить комментарий и макрос пункта, переместить пункт меню на другое место в иерархии.

Реализацию программ работы с меню мы рассмотрим в *части III*. На уровне меню AutoCAD дерево, показанное на иллюстрации, вызвано с помощью LISP-выражения

```
(ru-xml-pop-mnu "AR" "Спецрисунки строительные")
```

Функция `ru-xml-pop-mnu` вызывает программу просмотра меню, описанного в файле ar.xml (о том, что этот файл "живет" в каталоге %ruCADAllUsers%\XML\Menu, функция сама знает). Давайте посмотрим этот файл в текстовом редакторе (листинг 8.5).

Листинг 8.5. Файл ar.xml

```
<?xml version="1.0" encoding="windows-1251"?>
<root name="Архитектурно-строительные" sdata="">
<include href="ar_plan.xml" select="/*/*" sdata="Элементы планов"/>
<include href="ar_razr.xml" select="/*/*" sdata="Элементы разрезов"/>
<include href="ar_fasad.xml" select="/*/*" sdata="Элементы фасадов"/>
<include href="ar_sxkon.xml" select="/*/*" sdata="Схемы конструкций"/>
<include href="ar_kgb.xml" select="/*/*" sdata="Конструкции железобетонные"/>
<include href="ar_km.xml" select="/*/*" sdata="Конструкции металлические"/>
<include href="ar_kd.xml" select="/*/*" sdata="Конструкции деревянные"/>
```

```
<include href="ar_soed.xml" select="/*/*" sdata="Соединения конструкций"/>
<include href="tabl_ar.xml" select="/*/*" sdata="Таблицы АР"/>
</root>
```

Даже ничего не зная о языке XML, мы начинаем что-то понимать. Очевидно, здесь описаны ссылки на вложенные файлы.

А теперь "выполним" этот файл, сделав по нему двойной щелчок. Файл загрузится браузером Internet Explorer (рис. 8.2).

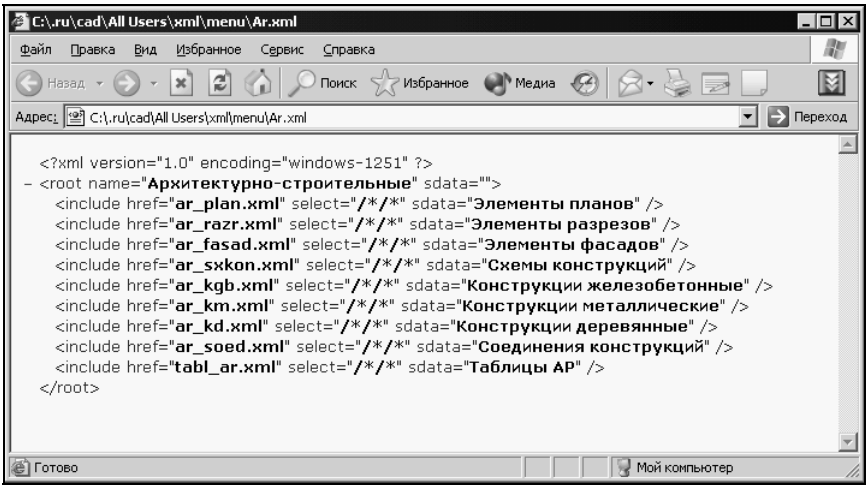


Рис. 8.2. Просмотр файла ar.xml в Internet Explorer

Щелкнув по минусу возле <root>, мы сворачиваем список в одну строчку!

Теперь загрузим в текстовый редактор файл ar_plan.xml (листинг 8.6) (в листинге мы оставили только те пункты, которые видны на иллюстрации).

Листинг 8.6. Фрагмент файла ar_plan.xml

```
<?xml version="1.0" encoding="windows-1251"?>
<root name="Элементы планов" sdata="">
<item name="Оси здания" image="AR\OSI_ZD.GIF" comment=
"Рисование координационных осей здания со множеством опций" sdata=
"(ru-app-load &quot;ru_ar_laying_axis&quot;)"/>
<item name="Категории помещений" sdata="">
<item name="Категория производства" image="AR\KAT_PRO.GIF" comment="Множественная
вставка изображения с автомасштабированием" sdata="(progn(ru-app-load
&quot;ru_block_insert_lib&quot;)
(start &quot;000000ar&quot; &quot;
КАТЕГОРИЯ_ПРОИЗВОДСТВА&quot; 1 1 1 NIL))"/>
<include href="e_obozn.xml" select="/*/*" sdata="Классы по ПУЭ"/>
</item>
<item name="Стены и перегородки" sdata="">
<item name="Стена" image="AR\WALL_2.GIF" comment=
"Рисование стены двойной полилинией со множеством опций" sdata="(progn(ru-app-load
&quot;ru_draw_dline&quot;)
```

```
(START &quot;стены&quot; NIL (strcat &quot;stena_&quot;
 (ru-unit-name))))"/>
<item name="Перегородка щитовая" image="AR\WALL_SH.GIF"
comment="Рисование перегородок щитовых специальным типом линии"
sdata="(progn(ru-app-load &quot;ru_draw_trass_line&quot;)
(start &quot;перегородки (штрихи слева)&quot; &quot;Забор&quot;
 (ru-lw-current-line-width) NIL NIL))"/>
<item name="Перегородка из стеклоблоков" image="AR\WALL_ST.GIF"
comment="Рисование перегородок из стеклоблоков стандартным обозначением"
sdata="(progn(ru-app-load &quot;ru_topo_fence_stone_hight&quot;)
(start 3))"/>
<item name="Ниша в стене" image="AR\WALL_NI.GIF" comment=
"Врезка ниши заданных размеров в отрезок или полилинию" sdata=
"(progn(ru-app-load &quot;ru_ar_union_pilaster_in_line&quot;)
(start &quot;ниши&quot; NIL))"/>
<item name="Пилястра" image="AR\WALL_PI.GIF" comment=
"Врезка пилястры заданных размеров в отрезок или полилинию"
sdata="(progn(ru-app-load &quot;ru_ar_union_pilaster_in_line&quot;)
(start &quot;пилястры&quot; T))"/>
</item>
<item name="Проемы" sdata="">
<item name="Окна" image="AR\OKNO_P.GIF" comment=
"Оконные проемы на плане с выбором размера, с четвертями и без четвертей"
sdata="(ru-app-load &quot;ru_ar_window_opening&quot;)"/>
<item name="Проем" image="AR\PROEM.GIF" comment=
"Врезка проема в нарисованную стену" sdata=
"(ru-app-load &quot;ru_ar_opening_in_wall&quot;)"/>
<item name="Двери" sdata="">
<item name="Однопольная" image="AR\DVER1.GIF" comment=
"Рисование полотна двери с заданным размером" sdata=
"(progn(ru-app-load &quot;ru_ar_door&quot;)(start 1))"/>
<item name="Двупольная" image="AR\DVER2.GIF" comment=
"Рисование полотна двери с заданным размером" sdata=
"(progn(ru-app-load &quot;ru_ar_door&quot;)(start 2))"/>
<item name="Складчатая" image="AR\DVER3.GIF" comment=
"Рисование полотна двери с заданным размером" sdata=
"(progn(ru-app-load &quot;ru_ar_door&quot;)(start 3))"/>
<item name="Раздвижная" image="AR\DVER4.GIF" comment=
"Рисование полотна двери с заданным размером" sdata=
"(progn(ru-app-load &quot;ru_ar_door&quot;)(start 4))"/>
<item name="Откатная" image="AR\DVER5.GIF" comment=
"Рисование полотна двери с заданным размером" sdata=
"(progn(ru-app-load &quot;ru_ar_door&quot;)(start 5))"/>
</item>
</item>
<item name="Колонны и опоры" sdata="">
<item name="Колонны ..." image="AR\KOLONN.GIF" comment=
"Рисование колонн различных типов с заданием шага и количества колонн
 в ряду" sdata="(ru-app-load &quot;ru_ar_column_row&quot;)"/>
<item name="Стойки безмасштабные" image="AR\STOIKA.GIF" comment=
"Условные обозначения стальных стоек" sdata=
"(ru-app-load &quot;ru_topo_pole&quot;)"/>
<item name="Опора металлическая" image="AR\OPOR_M.GIF" comment=
"Условное обозначение опоры с 4 колоннами" sdata=
"(progn(ru-app-load &quot;ru_topo_support&quot;)
(START &quot;000000to&quot; &quot;ЧЕРНЫЙ_КВАДРАТ_ТОПО&quot;))"/>
</item>
```

```
<item name="Антресоли и встройки" sdata="">
<item name="Антресоль" image="AR\ANTRES.GIF" comment=
"Условное обозначение антресоли заданных размеров" sdata=
"(progn(ru-app-load &quot;ru_topo_house_shed&quot;)
(START &quot;divide2&quot;))"/>
</item>
</root>
```

Загрузим и этот файл в браузер (рис. 8.3) и пощелкаем по плюсам и минусам. Мы видим, что файл "живет", его ветви сворачиваются и разворачиваются.

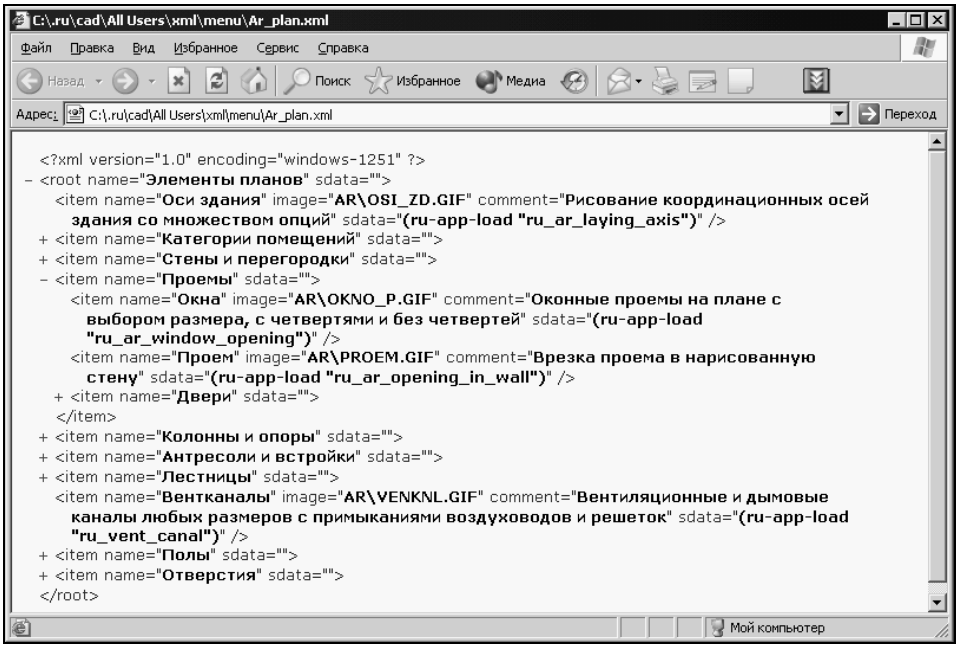


Рис. 8.3. Просмотр файла ar_plan.xml в Internet Explorer

Проанализировав и этот файл, сравнивая с его визуальным отображением на рисунке, мы, только с помощью рассуждений, узнаем многое о формате XML.

Вывод 1. Строка

```
<?xml version="1.0" encoding="windows-1251"?>
```

очевидно, является служебным заголовком, указывающим на версию языка и кодовую страницу (знаки вопроса явно служебные).

Вывод 2. Строка

```
<root name="Элементы планов" sdata="">
```

очевидно, обозначает "корень" дерева, а строка

```
<item name="Категории помещений" sdata="">
```

явно обозначает "ветку" дерева. Что такое `sdata`, пока непонятно.

Вывод 3. `</item>` и `</root>` очень похожи на завершение веток и всего дерева.

Вывод 4. Строка

```
<include href="e_obozn.xml" select="/*/*" sdata="Классы по ПУЭ"/>
```

очевидно, обозначает вложение другого файла, причем `href` наверняка указывает на имя этого файла, `sdata` похоже на отображаемое имя вложенного пункта, а `select="/*/*"` — не иначе, как указание выбрать все.

Вывод 5. Строка

```
<item name="Оси здания" image="AR\OSI_ZD.GIF" comment=
"Рисование координационных осей здания со множеством опций" sdata=
"(ru-app-load &quot;ru_ar_laying_axis&quot;)"/>
```

наверняка описывает исполняемый пункт меню. Все стало понятно:

- `name` — это название пункта, отображаемое в дереве;
- `image` — это относительное (по отношению к какому-то каталогу) имя файла, показываемого в качестве иллюстрации;
- `comment` — длинный комментарий к пункту меню;
- `sdata` — макроопределение пункта меню (теперь понятно, что для корня и ветвей макроопределений просто нет). Есть подозрение, что `&quot;` заменяет кавычки (наверное, внутри строки кавычки применять нельзя). И вообще все это очень похоже на язык разметки Web-страниц HTML.

Значит, можно написать и свое меню:

```
<?xml version="1.0" encoding="windows-1251"?>
<root name="Мое меню" sdata="">
        <item name="Раздел 1" sdata="">
                <item name="Пункт 1" image="AR\OSI_ZD.GIF" comment=
        "Сейчас выскочит птичка" sdata="(alert &quot;Ку-Ку!&quot;)"/>
                <item name="Пункт 2" image="AR\OSI_ZD.GIF" comment=
        "Сейчас выскочит другая птичка" sdata=
        "(alert &quot;Ку-Ка-Ре-Ку!&quot;)"/>
        </item>
        <item name="Раздел 2" sdata="">
                <item name="Подраздел 2-1" sdata="">
                        <item name="Пункт 1" image="" comment=
                "Это без иллюстрации " sdata=
                "(princ &quot; Пункт 1!&quot;)"/>
                </item>
        </item>
</root>
```

Что же, если вы, наши читатели, рассуждали подобным образом, то вас можно поздравить — вы самостоятельно раскрыли если не все секреты языка XML, то секреты меню ruCAD процентов на 90.

Мы также установили, что XML-файлы могут просматриваться браузером Internet Explorer, но в диалоговом окне ruCAD они отображаются гораздо лучше, очевидно потому, что эта программа знает о том, как правильно отображать такой формат.

Итак, не прочитав ни строчки теории, мы начинаем подозревать, что в файлах формата XML "что-то этакое есть". На этом пока и ограничимся. Пока запомним, что этот формат мы, даже без всяких познаний в теории, можем использовать для построения замечательных иллюстрированных меню.

Какие программы потребуются для реализации интерфейса

Для реализации задуманной концепции интерфейса потребуется разработать изрядное количество программ и служебных функций. Реализовывать их необходимо в первую очередь для того, чтобы сразу начинать работу с собственным меню.

Вот основные функции для работы с меню, которые реализовать очень просто:

- перегрузка и перекомпилирование главного меню;
- замена в разделе `POP6` меню специальных рисунков на заданное;
- вывод оглавления специальных меню с отметкой меню рабочей группы;
- загрузка и выгрузка файла фрагментного меню.

Для управления доступом к пунктам меню потребуются более сложные функции. Так как доступ к меню определяется правами пользователей, необходима разработка системы учета и проверки прав пользователей, а это уже гораздо серьезнее. Реестр пользователей мы будем вести в базе данных формата Microsoft Access, доступ к базе данных реализуем с помощью технологии ADO (ActiveX Data Objects), напрямую из LISP-программ. Такие функции нам пригодятся для многих целей.

Управление доступом к пунктам меню должно реализовываться с помощью целой библиотеки функций:

- запрет или разрешение доступа к пункту меню;
- пометка пункта меню "галочкой";
- пометка галочкой пункта "своей" рабочей группы в сборнике спецменю;
- снятие пометок с пунктов меню;
- управление доступом ко всем пунктам меню по шаблону тегов;
- пометки и снятие пометок всех пунктов меню по шаблону.

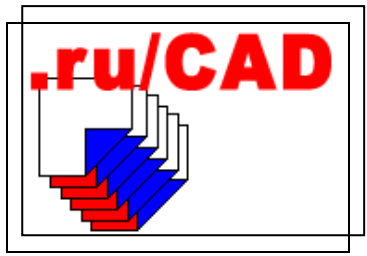
Внутри этих специализированных функций потребуется много служебных функций, на которых сейчас останавливаться не будем.

Помимо работы со стандартными элементами интерфейса нам потребуется реализация функций для работы с иллюстрированным деревом меню. Очевидно, что реализация графического интерфейса дерева на уровне Visual LISP невозможна. Это мы будем делать на Delphi и посвятим этому вопросу специальную *главу 16*.

Однако придется разработать и на Visual LISP много функций, решающих следующие задачи:

- вызов диалогового окна дерева меню и получение результатов;
- синтаксический разбор макросов дерева меню и их выполнение;
- получение и использование данных от обработчика меню.

ЧАСТЬ III



Разработка инструментальных средств для прикладного программирования

ГЛАВА 9

Инструменты разработчика в AutoCAD

Система AutoCAD обладает одним из самых лучших наборов инструментальных средств для разработки приложений, и это одна из главных причин ее феноменального успеха. Можно даже сказать, что "голая" система AutoCAD — это система не столько для пользователей, сколько для разработчиков. Возможности разработки приложений для AutoCAD растут с каждой новой версией.

Краткий обзор инструментальных средств программирования

Примечание

В книге Денниса Джампа[1] (или по крайней мере в ее русском издании[2]) утверждалось: "Настоящие программисты используют DXF-файлы (и только садисты работают с файлами чертежей — DWG-файлами)". Эта книга наглядно демонстрировала узкопрофессиональный подход высококвалифицированного программиста к решению задач САПР — в ней был отражен очень ценный опыт по сложному программированию работы с DXF на языке C, но обычные инженеры, читая эту книгу, удивлялись, каким сложным путем автор предлагал решать задачи, легко реализуемые с помощью языка AutoLISP, уже тогда имевшегося в AutoCAD.

Поддержка языка AutoLISP появилась в AutoCAD между версиями Release 6 (версия AutoCAD 2.1) и Release 7 (версия AutoCAD 2.5), т. е. в июне 1986 года. Это был первый мощный, но очень простой в освоении инструмент разработчика. Обычный инженер, освоивший работу в AutoCAD, мог без особого труда освоить приемы программирования на AutoLISP и получать приличные результаты. Решающее значение при этом имела функция `command`, которой в качестве параметров достаточно было передать имя команды и последовательность переменных или опций команды. Имена и опции команд инженер знал по "ручной" работе, а значения переменных (например, точек для рисования примитивов) легко вычислялись с помощью превосходных стандартных функций. По мере перехода к более сложным программам раз-

[1] Dennis N. Jump. AutoCAD Programming: TAB BOOKS Inc., 1989.

[2] Джамп Д. AutoCAD. Программирование: Пер. с англ. — М.: Радио и связь, 1992.

работчик постигал и красоту языка, особенно таких функций, как `eval`, `lambda`, `mapcar`, начинал понимать изящество представления свойств примитивов в виде ассоциированных списков, в которых с уникальным численным ключом свойства соотносилось значение свойства.

Разработчики из СССР в первые годы развития AutoCAD написали немало прикладных программ, многие из которых продолжают развиваться на качественно новом уровне. Были и очень интересные инструментальные средства для разработчиков.

Примечание

Для AutoCAD R10 была разработана система GLISP — автоматический генератор параметрических LISP-программ. Процесс создания программы заключался в разработке эскиза объекта средствами AutoCAD, нанесении на эскиз размеров и управляющих графических символов (параметров, констант и формул) и генерирования программы по эскизу. В полученную таким образом программу можно было передавать параметры из командной строки, из диалогового окна, из текстового файла и из динамической таблицы СУБД ABASE. GLISP был сделан мастерски, но эта технология не прижилась, т. к. оказалось, что проще и быстрее изучить сам AutoLISP и разрабатывать любые программы, чем освоить технологию подготовки эскизов.

Наблюдая за бесплодными спорами молодых приверженцев того или иного из императивных языков (C, Pascal или Basic) о том, какой из них "лучше", профессионалы посмеиваются и с успехом применяют любой, более подходящий для решения конкретных задач. Мнения разделяются скорее о системах программирования. Но язык LISP (LISt Processing — обработка списков) стоит особняком именно на языковом уровне.

LISP, как следует из его названия, предназначен для обработки списков, состоящих из "атомов" — абстрактных элементов, представляющих из себя формально неограниченные по длине цепочки символов. Атомы могут трактоваться и как строки, и как числа, или представлять собой некие логические структуры с вложенными на неограниченную глубину подсписками в виде иерархических деревьев. Для обработки списков используется функциональная модель, базирующаяся на теории lambda-исчислений Черча. LISP-программа является набором функций (также являющихся списками), при этом работа со списками осуществляется с помощью базового набора функций `car`, `cdr`, `cons`, `atom`, `eq`, `equal`. Благодаря рекурсивной системе обработки информации LISP позволяет очень компактно описывать функции, для реализации которых на других языках программирования потребовались бы сотни и тысячи строк кода.

Функции в LISP могут служить в качестве данных, подставляемых на место других аргументов. В результате каждого действия возникает некое значение, становящееся аргументом следующих действий.

Язык LISP был создан еще в 1956 году Джоном Маккарти (John McCarthy). С тех пор появилось много "диалектов" этого языка — BBNLisp, Interlisp, MacLisp, NIL (New Implementation of Lisp), Franz Lisp, Scheme, Flavors, LOOPS (Lisp Object Oriented Programming System), SPICE-Lisp, PSL (Portable Standard Lisp), Common LISP, который стал стандартом де-факто. В 70-х и начале 80-х годов XX века даже создавались специальные LISP-машины (Lisp Machines) в виде зашитой в ПЗУ программы в ряде компьютеров фирм Xerox и Texas Instruments. На LISP писали приложения операционной системы ITS (Incompatible Timesharing System) для мини-

компьютеров PDP-10. Существует мнение, что "перепиши хакеры из MIT, Стэнфорда и Карнеги-Меллонского университета ITS на LISP, вся история пошла бы в другую сторону, и открытыми системами мы называли бы, скорее всего, не потомков UNIX, а потомков ITS".

AutoLISP, включивший в себя специфичные функции для работы с AutoCAD, базируется на "диалекте" XLISP. Вот краткая хронология развития AutoLISP:

- Release 7 (Version 2.5) — введен доступ к графической базе данных рисунка;
- Release 8 (Version 2.6) — появились 3D-точки, функции `getcorner`, `getkword` и `initget`;
- Release 9 — появилась функция `command` и фильтры для `ssget`;
- Release 10 — введены расширения функций `findfile` и `load`, новые функции `getenv`, `handent` и `vports`;
- Release 11 — введены расширения функции `entget`, новые функции `cvunits` и `entmake`;
- Release 12 — введены расширения функций `ssget` и `initget`, появились новые функции `alert`, `getfiled` и `textbox`, а также возможность создания диалоговых окон с использованием языка DCL (Dialog Control Language);
- Release 13 — введены новые функции `acad_strlsort`, `autoload`, `help` и `acad_colordlg`;
- Release 14 — появилась среда разработки Visual LISP, поставляемая пока как отдельный продукт, и возможность использования механизмов ActiveX;
- 2000 (Release 15) — Visual LISP встроен в AutoCAD, появилось множество новых полезных функций (сохранились и все старые), функции, определяемые через `defun`, стали компилированными, но, самое главное, окончательно внедрена работа с любыми COM-объектами через механизмы ActiveX.

Visual LISP является потомком программного продукта Vital Lisp, права на который были приобретены фирмой Autodesk у Basis Software. Многие программисты до сих пор используют Vital Lisp. Для AutoCAD R14 Visual Lisp поставлялся как отдельное дополнительное расширение, но, начиная с версии AutoCAD 2000, усовершенствованный Visual LISP стал неотъемлемой частью системы AutoCAD.

Помимо LISP фирма Autodesk постоянно внедряет и другие средства для разработки приложений. При всей красоте LISP программистов-профессионалов, работающих на этом языке, относительно мало. Да и откуда им взяться, если в Америке 90% программ для Windows пишут на Visual Basic! Необычный синтаксис (скобочная префиксная запись) стал одной из причин сравнительно редкого использования LISP и его потомков в наши дни. Возможно, что отрицательный эффект дает позиционирование LISP как языка "искусственного интеллекта".

Примечание

При наблюдении за политикой фирмы Autodesk в области средств разработки создается впечатление, что там происходит какая-то внутренняя борьба между различными группировками, а "правая рука не знает, что делает левая". Например, старший программист Autodesk Роберт Вениг в 1989 году на вопрос: "А как насчет компилятора LISP?" — отвечал: "По поводу компилятора LISP мне нечего сказать[1]", хотя в 1989 году уже выпускался

[1] Мир САПР. Пробный выпуск 1990 г. Приложение к журналу Мир ПК. Стр. 43.

(с Copyright 1989 Autodesk Ltd) ACOMP — великолепный компилятор программ, написанных на языке AutoLISP. Компилятор распространялся СП "ПАРАЛЛЕЛЬ", прилагался бесплатно к русской версии AutoCAD R10 и за 250 фт. стрл. к английской версии[1]. В том же интервью Роберт Вениг много говорил про "Си-связки", про склонность к OS/2 "из-за того, что на нее делает упор компания Microsoft". Чем этот "упор" закончился, мы теперь знаем.

Для привлечения к разработке приложений для AutoCAD дополнительных сил фирма Autodesk задумала и реализовала новую систему разработки приложений.

В 1989 году в AutoCAD R11 появляется *ADS (AutoCAD Development System)* — система разработки расширений AutoCAD на языке C. Это и были "Си-связки", которые дополняли AutoLISP необходимыми функциями. Сначала для разработки ADS-расширений можно было использовать различные компиляторы языка C.

Впоследствии система ADS была заменена на расширение *ARX (AutoCAD Runtime Extension)*, предоставляющее полностью объектно-ориентированный интерфейс между программами, разработанными на C++, и AutoCAD. Название Runtime Extension подразумевает одно из ключевых отличий от предшественницы ARX, а именно ADS, приложения которой работали как внешние процессы и общались с AutoCAD при помощи взаимодействия процессов (Interprocess Communication). ARX-приложение работает в том же адресном пространстве, что и AutoCAD, как динамически связываемая библиотека (DLL) и имеет прямой доступ к графической базе данных AutoCAD. Важнейшая особенность ARX заключается в том, что появилась возможность расширения AutoCAD не только за счет новых команд, но также за счет новых типов примитивов.

В 1995 году в AutoCAD R13 появляется возможность использования библиотеки *ObjectARX*, постоянно развивающейся и дополняющейся по мере выпуска новых версий AutoCAD. Про альтернативные среды разработки теперь пришлось забыть — ARX-приложения можно было разрабатывать только с использованием Microsoft Visual C++.

Появление ADS и ARX дало возможность разрабатывать очень мощные приложения к AutoCAD, такие как Autodesk Architectural Desktop (ADT), но разрушило стройную и красивую систему внутреннего представления данных примитивов AutoCAD в виде ассоциированных списков. Списки-то остались, но C-программисты испортили всю логику, сделав допустимыми совпадающие ключи. С тех пор и началась утеря лучших приемов программирования на LISP.

В 1996 году начинается внедрение технологии *ActiveX Automation* — стандартного метода Microsoft предоставления доступа к частям одного приложения для применения другими приложениями. В 1997 году для использования всех потенциалов ActiveX в AutoCAD вводится возможность разработки приложения на *VBA (Visual Basic for Applications)*. Язык VBA базируется на стандартном Visual Basic и, по идее Microsoft, может внедряться в любые приложения. Используя VBA, можно создавать программы, управляющие теми частями основного приложения, доступ к которым предоставлен через ActiveX Automation.

Вообще-то ActiveX Automation позволяет создавать приложения на любом языке, который поддерживает интерфейс ActiveX Automation: C++, Delphi, Visual Basic. Все это отдельные среды, не интегрированные в AutoCAD. VBA встроен в среду

[1] Мир САПР. Выпуск 1/1991 г. Приложение к журналу Мир ПК. Стр. 1.

AutoCAD (конечно, не бесплатно) и якобы нет необходимости приобретать дополнительные программные средства — Microsoft свою долю с вас уже получила.

В 1998 году новая среда разработки LISP-программ Visual LISP также получает возможность использования механизмов ActiveX. Это открывает новые возможности, недоступные в добром старом AutoLISP.

Сравнение AutoLISP и Visual LISP

На наш взгляд, в названии Visual LISP первое слово кажется не очень корректным рекламным трюком. Никаких средств "вижуального" конструирования программ, как в других Visual-системах (C++, FoxPro, Basic и т. п.), кроме предварительного просмотра написанных вручную диалоговых окон, в Visual LISP нет. Имеются прекрасный редактор с подсветкой синтаксиса, средства для отладки и компилятор. Но все это обязательные компоненты любой интегрированной среды разработчика (IDE), имевшиеся, на таком уровне, во многих системах очень давно. Хорошую современную IDE сделать не просто, и фирме Autodesk это явно не по силам.

Многие разработчики ожидали, что Visual LISP станет по-настоящему "вижуальным", но фирма Autodesk пошла иным путем, т. е. начала внедрять в AutoCAD VBA и, кажется, охладела к доброму старому LISP. На наш взгляд, это неверное решение. Во многом благодаря такому доступному для разработчиков средству, как AutoLISP, стал популярен и сам AutoCAD. В мире наработаны десятки тысяч LISP-программ для AutoCAD, выросло целое поколение LISP-программистов и, если они в конце концов лишатся этой среды, AutoCAD может потерять популярность. Не исключено, что в результате он вообще превратится в "Microsoft CAD".

Однако оставим в стороне эти домыслы и поближе познакомимся с различиями между Visual LISP[1] и AutoLISP.

Интегрированную среду разработчика пока рассматривать не будем. В AutoLISP ее просто нет.

Во-первых, Visual LISP поддерживает (с некоторыми нюансами, которые будут рассмотрены далее) все функции AutoLISP. Это означает, что все старые программы должны работать и в "новых" системах AutoCAD. Так считает фирма Autodesk. Да, простенькие программы работать будут, а вот сложные, с модификацией кода — вряд ли.

Во-вторых, в Visual LISP единого представления функций и данных в виде списков больше нет! AutoLISP, начиная с версии 2000, больше не является "натуральным", т. е. не работает со списками кода, которые могли объединяться во время работы. Теперь LISP-программы всегда выполняются в откомпилированном виде, даже если они загружались из LSP-файлов или вообще вводились в командной строке. Компилирование происходит всегда и в памяти. Хотите убедиться? Наберите в командной строке AutoCAD версии 2000 и старше текст тестовой функции:

```
(defun test (msg)(princ msg))
```

---

[1] Строго говоря, Visual LISP — это среда разработки, а язык по-прежнему называется AutoLISP (см., например, пункт меню **Tools | AutoLISP** (Средства | AutoLISP)). Однако в среде программистов привился термин "Visual LISP" по отношению к развитию языка за счет добавления большого количества полезных функций с префиксами `vl-`, `vla-`, `vlax-`, `vlr-`. Новые функции резко расширили возможности разработчиков.

а потом попытайтесь вывести текст этой функции на экран, набрав

```
!test
Получим ответ: #<SUBR @01b32f14 TEST>.
```

Если бы мы провели такой же эксперимент в AutoCAD R14, то получили бы

```
((MSG) (PRINC MSG))
```

Это означает, что в AutoCAD 2000 и старше (со встроенным Visual LISP), функции, определенные через `defun`, перестали быть списками и их нельзя программно модифицировать. Для того чтобы не утерять такой важной особенности LISP, была введена функция `defun-q`. Повторяем эксперимент:

```
(defun-q test (msg)(princ msg))
!test
Результат: ((MSG) (PRINC MSG))
```

Мы получили функцию в виде списка. Для того чтобы избежать немедленной трансляции, следует определять некоторые функции именно через `defun-q`. Это может пригодиться для функций, чаще всего модифицируемых во время работы программ — `s::startup` и `*error*`.

В третьих, в Visual LISP появилось много новых функций, которых так не хватало в AutoLISP — работа с файлами, реестром, сортировка, дополнительные функции для работы со списками и т. п. Специально мы их разбирать не будем, применение таких функций очевидно.

В четвертых, и это самое главное, Visual LISP поддерживает технологию ActiveX Automation, разработанную и внедряемую фирмой Microsoft. Вот за это можно простить фирме Autodesk все! Или почти все.

При работе по технологии ActiveX одни приложения (клиенты) имеют доступ к объектам других приложений (серверов) и могут этими объектами управлять. Самым важным приложением-сервером для программ, написанных на Visual LISP (далее VL-программ), является AutoCAD. Для доступа к объектам AutoCAD в Visual LISP включено очень много функций (все они имеют префикс `vla-` или `vlax-`).

Однако из VL-программ можно получить доступ и к объектам других приложений. Обычно в пример приводят программы Microsoft Office, но такие программы вы можете написать и сами. Найдите на компьютере все файлы с расширениями tlb и olb и вы получите впечатляющий, но неполный список серверов, к объектам которых вы имеете доступ! А в ветвях реестра HKEY_CLASSES_ROOT*\Clsid вы найдете полный список серверов автоматизации, которые только и мечтают, чтобы к ним обратились из VL-программ.

Здесь мы рассмотрели принципиальные отличия, в следующих главах будем более детально сопоставлять приемы работы.

Когда можно и нужно использовать VBA

С внедрением ActiveX Automation в AutoCAD была введена и поддержка Visual Basic for Applications (VBA), включающего язык и среду программирования. Сначала VBA использовался в приложениях Microsoft Office и был там очень к месту, удачно за-

менив Word Basic и став единым стандартным средством разработки. Внедрение VBA в систему AutoCAD, имеющую такие мощные среды разработки, как LISP и ADS/ARX, объясняется только популизмом и стремлением делать деньги вместе с Microsoft. Visual Basic и VBA, как его разновидность, очень популярны в США. Обычно уверяют, что:

- с Visual LISP знакомо немного программистов;
- т. к. Basic был разработан как язык для начинающих, а VBA продолжает эту традицию, то "если вы изучите несколько простых программных концепций, вы сами сможете настраивать AutoCAD".

Сами любители VBA обычно боятся LISP потому, что в нем "загадочные команды и всюду круглые скобки". Все это несерьезные аргументы. Давайте сами поищем более убедительные.

Преимущества VBA

Во-первых, VBA, в отличие от Visual LISP, действительно "вижуальный", т. е. обладает приличными средствами для конструирования диалоговых окон и интегрированной средой, облегчающей программирование и отладку. При конструировании диалоговых окон в VBA можно использовать элементы управления OCX, получившие большое распространение. Это серьезный аргумент.

Во-вторых, VBA сильно интегрирован с Windows. В VBA имеются собственные функции, отсутствующие в Visual LISP. Впрочем, многие встроенные функции на самом деле работают через ActiveX с другими COM-серверами, т. е. через цепочку посредников.

В-третьих, из VBA можно загружать DLL-библиотеки и выполнять определенные в них функции. Из Visual LISP этого делать напрямую нельзя. Возможно, что это главное преимущество VBA, на которое редко обращают внимание.

В-четвертых, в справочной системе AutoCAD все примеры по использованию ActiveX написаны на VBA и начинающий программист сразу может их использовать. Для Visual LISP код примеров придется переводить на другой язык.

Примечание

К числу сомнительных достоинств VBA можно отнести возможность внедрения проекта в DWG-файл. Это, на первый взгляд, несомненное достоинство может оказаться крупным недостатком, т. к. внедренный в файл проект является "открытой дверью" для вирусов. Вирусы, внедренные в документы Microsoft Office, распространились очень широко, т. к. для написания их не нужно никакой квалификации, а деструктивных личностей в мире больше, чем достаточно. До DWG-файлов эта эпидемия пока не дошла, но это только вопрос времени.

Недостатки VBA

Приложение VBA взаимодействует с AutoCAD только через ActiveX Automation, а эта технология имеет свои пределы. Какие объекты, их свойства и методы предоставили разработчики AutoCAD, с такими, и только такими объектами можно работать через ActiveX. Возможности ActiveX все-таки ограничены (это относится и к ис-

пользованию их в Visual LISP). Попробуйте, например, методами ActiveX нарисовать в интерактивном режиме какую-нибудь трассу, примерно так, как работает команда PLINE, и все станет понятно.

Приложение VBA не может вообще взаимодействовать с AutoLISP. Подпрограммы с ActiveX Automation не могут вызывать подпрограммы на AutoLISP, так же они не могут устанавливать значения переменных AutoLISP. Можно, используя метод `SendCommand`, отправить в командную строку AutoCAD вызов LISP-функции, но результат работы этой функции получить нельзя. VBA-программисты, думающие, что им это и не надо, глубоко заблуждаются.

Приложение VBA не может взаимодействовать и с другими подобными приложениями. В макрос, написанный на VBA, нельзя передать параметры. Обмен данными возможен только или через файлы, или через системные переменные AutoCAD, или через пользовательские словари DWG-файлов.

Все эти недостатки сильно ограничивают область применения VBA в крупных системах, а именно такую систему мы и разрабатываем. Но для решения отдельных изолированных задач приложения VBA можно применять с успехом. Там, где это будет рационально, мы и будем использовать VBA-проекты. Хотя бы для примера.

Читателям, делающим ставку только на VBA, рекомендуем ознакомиться с грядущей технологией Microsoft .NET, например, по книге Дэвида Платта[1], в которой он не стесняется критиковать "решения партии", хотя книга выпущена издательством Microsoft Press. Если не рассматривать технические детали и уверения, что все делается "на благо человека", можно понять, что истинной целью является сделать богатых еще богаче, а бедных еще беднее. Перехода на .NET и кончины платформы Win32, если уж Microsoft вложила в это столько денег, не миновать, но больше всех могут пострадать именно Basic-программисты.

Примечание

Почитайте на **www.vbnet.ru** статьи с "душераздирающими фразами":

"Но настал 2000 год. Именно в этом году Microsoft впервые объявила о планах по созданию программной платформы .NET, которая в конечном счете и привела к кончине Visual Basic, а с ним и духа Basic, созданного 10 лет назад".

"Что будет с огромной армией программистов, которые писали свои программы на языке Visual Basic?"

"VB6 вполне успешно протянет еще год-два. За это время Майкрософт даст откат от своей объектно-ориентированной политики в области бейсика. А уж если она этого не сделает — Borland Delphi[2] станет монополистом рынка программных продуктов этого класса для PC".

Это пишут настоящие знатоки Visual Basic. Наверняка в одной из ближайших версий AutoCAD все это коснется и любителей VBA.

---

[1] Платт Д. С. Знакомство с Microsoft .NET/ Пер. с англ. — М.: Издательско-торговый дом "Русская редакция", 2001.

[2] В то же время многие Delphi-программисты признали Visual Basic версии 7 "настоящим" объектно-ориентированным языком. Но и последняя, восьмая версия Delphi уже ориентирована на платформу .NET, а версия 7 стала последней для платформы Win32.

Как мы будем использовать ObjectARX

Пакет ObjectARX, как мы уже упоминали, является самым мощным средством разработки приложений для AutoCAD. В такой книге, как наша, описать технологию работы с ObjectARX просто невозможно, да и книга не является учебным пособием, а рассказывает, как разработать конкретную систему. В нашей системе ruCAD "настоящие" ARX-приложения и не потребуются. Но с использованием ObjectARX можно относительно просто создавать и библиотеки функций, используемых в LISP-программах. Несколько таких функций нам потребуются, и мы их разработаем. Этому вопросу посвящена *глава 14*.

Как мы будем использовать Delphi

Большое количество инженеров, не являющихся профессиональными программистами, овладели разработкой приложений в среде Borland Delphi. Многие из них одновременно работают с AutoCAD, программируют на LISP и хотели бы соединить эти системы разработки приложений. Фирма Autodesk официально не поддерживает Delphi в качестве стандарта для разработки приложений, но внедрение в AutoCAD технологий ActiveX делает его доступным для программ, созданных в средах, использующих *COM (Component Object Model)*.

COM — одна из основных технологий, на которой основывается Windows. В виде COM-интерфейсов реализован и API *(Application Programming Interface)* системы AutoCAD. И Visual LISP, и VBA взаимодействуют с AutoCAD через COM-интерфейсы, только это взаимодействие спрятано от пользователя, которому предоставляются более удобные высокоуровневые функции.

Модель COM предоставляет возможность создания многократно используемых компонентов, независимых от языка программирования. Такие компоненты называются *COM-серверами* и представляют собой исполнимые файлы (EXE) или динамические библиотеки (DLL), специальным образом оформленные для обеспечения возможности их универсального вызова из любой программы, написанной на поддерживающем COM языке программирования. При этом COM-сервер может выполняться как в адресном пространстве вызывающей программы (In-Proc-сервер), так и в виде самостоятельного процесса (Out-Of-Proc-сервер), или даже на другом компьютере (Distributed COM).

В книге "AutoCAD 2002" (серия "В подлиннике") мы приводили примеры совместного использования Delphi и AutoCAD, но это были отвлеченные примеры различных возможных решений (ARX, написанный на C++ с использованием функций из Delphi-DLL, ARX только на Delphi, рисование из Delphi-программы в AutoCAD и т. п.). Сейчас мы разрабатываем конкретную систему и должны решать соответствующие определенные задачи. В системе ruCAD мы, в основном, будем использовать Delphi для разработки COM-серверов в виде DLL, доступ к объектам которых будет осуществляться из LISP-программ. В соответствии с задачами, намеченными в предыдущих главах, нам потребуются следующие серверы автоматизации:

- работы с иллюстрированным деревом меню;
- работы с классификатором слоев;

- работы с реестром пользователей;
- диалогового редактирования координат полилиний;
- выбора файлов и папок.

Кроме того, по ходу разработки программ мы будем выяснять необходимость и реализовывать любые другие серверы, необходимость в которых возникнет.

Соглашения о кодировании

В *главе 1* мы говорили о необходимости выработки стандартов программирования. Одной из составных частей таких внутренних стандартов являются соглашения о кодировании, т. е. о правилах оформления исходных текстов программ. Официальных стандартов кодирования не существует, но есть стандарты де-факто, обычно это стили оформления, принятые у разработчика системы программирования. Например, существует стандарт оформления исходных текстов для Delphi, ознакомиться с которым можно на **Borland Community site**[1]. Дополненный перевод этого стандарта можно найти на сайте "Королевство Delphi"[2].

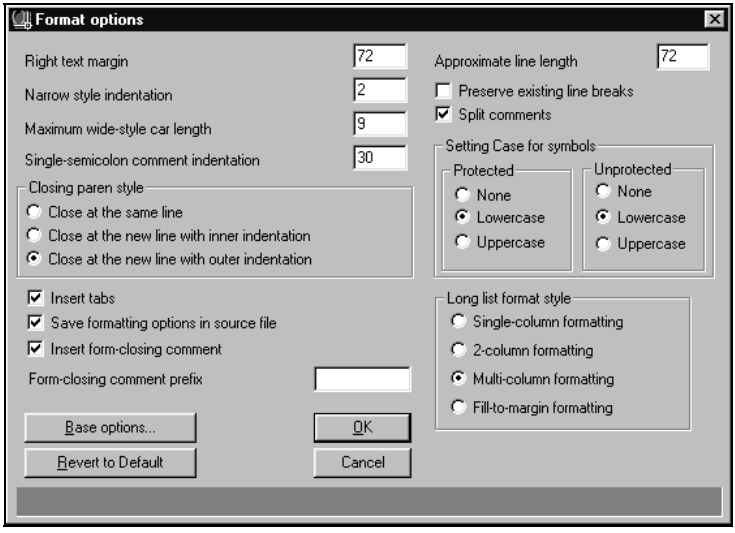


Рис. 9.1. Установка опций форматирования для Visual LISP

Основным языком программирования в нашей системе ruCAD является LISP, для него мы и выработаем свой стандарт кодирования. Для LISP также имеются несколько стилей кодирования, например "CanOfWorms AutoLISP Coding Conventions", "Emacs Lisp Coding Conventions" и др. Для нашей системы многое предопределено использованием редактора Visual LISP с его системой форматирования исходных

[1] **http://community.borland.com/soapbox/techvoyage/article/1,1795,10280,00.html**.

[2] **http://delphi.vitpc.com/article/coderules.htm**.

текстов. Определенные ограничения вносит необходимость придерживаться формата книги (длина строки ограничена 72 символами). Опции форматирования в Visual LISP устанавливают в диалоговом окне **Format options** (Опции форматирования), вызываемом из меню **Tools | Visual LISP Format options** (Сервис | Параметры форматирования Visual LISP) (рис. 9.1).

Такие параметры форматирования позволят нам оптимально размещать текст как для практической работы, так и для книги. Обратите внимание на то, что мы предусмотрели перевод в нижний регистр и защищенных, и собственных символов. Это означает, что все имена функций и переменных будут переводиться в нижний регистр.

Как добиться единообразного внешнего вида программ

Исходные тексты, имеющие единообразный внешний вид, всегда легче читать и обрабатывать другими программами. Программисты обычно вырабатывают собственный стиль оформления. Не столь важно, какой именно (многое зависит от личного вкуса), главное, чтобы оформление было единым. Мы будем соблюдать перечисленные ниже собственные правила оформления.

Имена функций

Имена всех функций ruCAD должны начинаться с префикса `ru-`. Это понадобится для того, чтобы отличать наши функции от стандартных и посторонних.

Имена "низкоуровневых" функций, не предназначенных для использования в конечных программах, должны начинаться с префикса `_ru-`. Знак подчеркивания в начале имени будет нам об этом напоминать. Такие функции мы не будем документировать, когда придет время распространения наших библиотек.

Имена глобальных переменных мы будем обрамлять символом `*` и начинать их с префикса `ru_`, например `*ru_root_dir*`.

Имена функций и переменных мы будем писать в нижнем регистре. А почему не в верхнем, и не в смешанном? Да потому, что:

- так нам нравится, мы привыкли к стилю языка C;
- рано или поздно имена функций нам придется использовать в качестве имен файлов на Web-сервере с операционной системой UNIX, а там регистр имеет большое значение, и при закачке файлов на такой сервер обычно имена файлов переводятся в нижний регистр.

После префикса указывается тематическая группа, к которой относится функция, а разделение имен на части для удобочитаемости производится *для имен функций символом "–", а для имен переменных символом "_"*. Смешанный регистр для разделения имени на части, как это делается обычно, мы не применяем потому, что если имеется возможность приведения имен к одному регистру при форматировании, то этой возможностью непременно кто-то воспользуется, а восстановить смешанный регистр невозможно. Кроме того, единый стандартный разделитель слогов позволит

разработать программу для быстрой навигации по тематическому дереву больших библиотек функций.

Примеры имен функций:

- `ru-conv-deg-to-rad` — функция из группы преобразований (не `ruConvDegToRad`, не `Dtr` и не `deg->rad`[1]);
- `ru-conv-rad-to-deg` — функция из группы преобразований;
- `ru-get-dist` — функция из группы ввода данных;
- `ru-get-point` — функция из группы ввода данных;
- `ru-user-get-rights-info` — функция из группы работы с пользователями;
- `_ru-user-db-connect-string` — функция из группы работы с пользователями (низкоуровневая);
- `ru-trass-draw-lw-first-prev-sample` — функция из группы рисования трасс;
- `ru-trass-draw-lw-from-pt2` — функция из группы рисования трасс;
- `ru-pline-draw-closed-lw-msg` — функция из группы рисования полилиний;
- `ru-pline-draw-closed-lw-msg-from-pt1` — функция из группы рисования полилиний.

Длинные, но ясные имена функций сделают программы более понятными[2]. Такие имена неудобно набирать вручную, и мы советуем никогда этого не делать, даже если имя короткое, и вы (как вам кажется) точно его помните. Сколько ошибок бывает из-за такой самоуверенности! Мы рекомендуем всегда копировать имена функций и переменных.

Примечание

На устаревшие, но еще встречающиеся рекомендации по ограничению длины имени функции шестью символами теперь можно не обращать внимания. Проблем с объемом памяти в LISP теперь нет[3].

Вложенные функции

Вложенные функции, определенные в теле основной функции, включаются в список локальных переменных. Имена вложенных функций, как правило, начинаются с символа подчеркивания и включают начало имени основной функции (листинг 9.1).

Листинг 9.1. Пример вложенных функций

```
(defun ru-get-string (params / _get-string-clear _get-string-upd   other_var)
 (defun _get-string-clear ()
```

[1] Наглядные имена функций с угловыми скобками очень неудобно будет записывать в XML-документы, которые мы в дальнейшем будем широко использовать. Кроме того, угловые скобки нельзя включить в имя LISP-файла.

[2] "Читабельные" имена позволят избежать излишних комментариев и в тексте книги.

[3] На самом деле граница объема есть, но достичь ее очень трудно.

```
;; Тело локальной функции
);_ конец локальной функции
 (defun _get-string-upd ()
;; Тело локальной функции
);_ конец локальной функции
;;; Тело главной функции
);_ конец главной функции
```

Имена команд AutoCAD

Имена команд и системных переменных AutoCAD записываются в верхнем регистре. Напоминаем *(см. главу 4)*, что перед именем команды следует указывать префикс "_.", а перед опциями команд префикс "_".

Использование комментариев

Редактор Visual LISP поддерживает несколько стилей комментариев:

- *;|Блочный комментарий|;* — включается в любое место исходного текста (даже на разных строках), при форматировании остается на месте;
- *;;;Комментарий после трех точек с запятой* — после применения форматирования сдвигается в первую позицию;
- *;;Комментарий после двух точек с запятой* — после применения форматирования сдвигается в позицию, соответствующую началу следующей строки;
- *;Комментарий после одной точки с запятой* — после применения форматирования сдвигается в позицию, заданную в параметре **Single-semicolon comment indentation** (Отступ комментария с одной точкой с запятой);
- *;_Послестрочный комментарий после одной точки с запятой и символа подчеркивания* — после применения форматирования сдвигается в позицию с отступом на один пробел от последнего символа строки, в конце которой стоит.

Листинг 9.2. Пример исходного текста до форматирования

```
(defun ru-menu-load-partial (menu_file_name group_name / tmp)
;;; Загрузка фрагментного меню
;;; Параметры:
;;; menu_file_name - имя файла фрагментного меню (без расширения)
;;; group_name  - имя группы в файле меню
;|
Пример:
Command: (ru-menu-load-partial "ruLib" "ruLib")
Menu loaded successfully. MENUGROUP: ruLib
Меню ruLib.mnu успешно загружено.
|;
(if (not (menugroup group_name))(progn (if(findfile (strcat menu_file_name
".mnu"))(progn (command "_.MENULOAD" menu_file_name)
;; Вычисление последнего имеющегося падающего меню
(menucmd (strcat "P" (itoa (1+ (_ru-menu-get-max-pulldown-number;|служебная
функция|;))) "=+" group_name ".POP1"))
; Информация для пользователя
(princ (strcat "\nМеню " menu_file_name ".mnu успешно загружено.")))
```

```
(progn (alert (strcat "Не могу найти " menu_file_name ".mnu")))))
;; Если такое меню уже загружено, известить пользователя
(princ (strcat "\nМеню " menu_file_name ".mnu уже загружено")))(princ))
```

Листинг 9.3. Пример исходного текста после форматирования

```
(defun ru-menu-load-partial (menu_file_name group_name / tmp)
;;; Загрузка фрагментного меню
;;; Параметры:
;;; menu_file_name - имя файла фрагментного меню (без расширения)
;;; group_name - имя группы в файле меню
;|
Пример:
Command: (ru-menu-load-partial "ruLib" "ruLib")
Menu loaded successfully. MENUGROUP: ruLib
Меню ruLib.mnu успешно загружено.
|;
 (if (not (menugroup group_name))
 (progn
  (if (findfile (strcat menu_file_name ".mnu"))
    progn
     (command "_.MENULOAD" menu_file_name)
    ;; Вычисление последнего имеющегося падающего меню
     (menucmd (strcat "P"
             (itoa (1+ (_ru-menu-get-max-pulldown-number
                 ;|служебная функция|;
                 )
                );_ 1+
            );_ itoa
             "=+"
             group_name
             ".POP1"
        );_ strcat
    )   ; Информация для пользователя
     (princ
     (strcat "\nМеню " menu_file_name ".mnu успешно загружено.")
    );_ princ
   );_ progn
    (progn (alert (strcat "Не могу найти " menu_file_name ".mnu")))
 );_ if
);_ progn
;; Если такое меню уже загружено, известить пользователя
 (princ (strcat "\nМеню " menu_file_name ".mnu уже загружено"))
);_ if
 (princ)
);_ defun
;|"Visual LISP© Format Options"
(72 2 30 2 T nil 72 9 1 1 0 T nil T T)
;*** DO NOT add text below the comment! ***|;
```

Комментарии к функциям мы тоже будем делать не так, как все. Комментарии мы будем размещать не до определения функции, а в теле функции, после объявления имени и локальных переменных. Такое расположение комментариев позволит при выделении функции в редакторе Visual LISP включать в выделенный фрагмент и комментарии и облегчит перенос функций.

Пример комментария к функции был приведен в листинге 9.3. Особое внимание мы будем уделять примерам вызова функций, которые должны заключаться в блочные комментарии. Эти примеры будут являться к тому же и тестами на правильность работы. Пример вызова должен быть написан так, чтобы его можно было скопировать в систему AutoCAD, выполнить и получить результат. Следовательно, в качестве параметров не должны использоваться имена переменных, а должно быть указано явное значение, возможно, в нескольких вариантах. Например:

```
(defun ru-match-is-even (number)
;;; Проверяет, четное ли число number
;|Примеры
Command:(ru-match-is-even 2)
T
Command: (ru-match-is-even 123)
nil
Command: (ru-match-is-even 0)
T
Command: (ru-match-is-even -1)
nil
Command: (ru-match-is-even -2)
T
|;
(zerop (rem number 2))
);_ defun
```

В книге, ради экономии места, мы будем упрощать комментарии — не будем комментировать очевидные вещи (все-таки это не учебник), но подробнее, чем в реальных исходных текстах, разъяснять сложные участки.

Замечание

Форматирование исходных текстов в книге может отличаться от приведенных правил. Вызвано это особенностями формата издания и стремлением авторов сократить место, занимаемое исходными текстами при слишком разреженном форматировании.

Как единообразно именовать файлы

Большое значение имеет и единая система именования файлов. Больше всего у нас будет файлов с исходными текстами на языке LISP. Здесь мы также будем придерживаться собственных правил.

Файлам конечных программ мы будем присваивать длинные, понятные имена с префиксом "ru_", указанием тематической группы и разделением слов символом подчеркивания, например:

- ru_block_insert_table.lsp (блок, вставка таблицы);

- ru_block_explode_on_layer.lsp (блок, расчленение на слой);
- ru_ar_face_door_residental.lsp (строительная часть, фасад, дверь жилого здания).

Исходные тексты конечных программ мы будем размещать в одном каталоге

```
%ruCADRootDir%\Source\Lisp
```

Откомпилированные файлы будут автоматически помещаться в каталог

```
%ruCADRootDir%\All Users\App
```

Если включен режим работы разработчика (определяемый наличием глобальной переменной `*ru_developer*`) и найден исходный файл, то при попытке загрузки программы будет автоматически выполняться компиляция и всегда будет загружаться последняя откомпилированная версия. Это позволит избежать вызова сотен операций компиляции вручную.

Каждую библиотечную функцию мы будем сохранять в отдельный файл[1], с таким же именем, как и имя функции. Файлы функций будут располагаться в каталоге %ruCADRootDir%\Source\Lisp\Lib и его подкаталогах. Компиляция функций будет выполняться в одну библиотеку.

Группы функций мы будем размещать в отдельных подкаталогах каталога Lib. Например, в каталоге %ruCADRootDir%\Source\Lisp\Lib\Get будут размещаться файлы ru-get-*.lsp, а в каталоге %ruCADRootDir%\Source\Lisp\Lib\String — файлы ru-string-*.lsp.

К такой системе мы пришли в результате многолетних экспериментов. При разработке большой системы удержать в памяти имена и параметры сотен функций просто невозможно. Постоянно требуется взглянуть на исходный текст функции. Поиск исходных текстов занимал массу времени. Мы помещали группы функций в сборники, но приходилось выяснять, в каком сборнике находится функция, просматривая их. При размещении всех функций в одном большом файле приходилось заниматься поиском в этом файле. Если же размещать каждую функцию в отдельный файл, в иерархии каталогов приходилось заниматься просмотром или поиском в каталогах, что немногим лучше. В результате пришлось сделать программу для просмотра дерева функций (рис. 9.2).

Примечание

Программа ruLispExplorer, сканируя каталог библиотеки, автоматически строит дерево файлов-функций. При движении по дереву файл загружается в редактор (знающий подсветку синтаксиса LISP), в котором можно выполнить небольшую правку без загрузки в Visual LISP, скопировать имена функций и примеры использования. Для каждой функции можно отредактировать ее краткое описание, сохраняющееся в файле dirinfo.ini и "на лету" создать иллюстрацию. Кроме того, можно экспортировать текст функции в HTML-файл с сохранением подсветки синтаксиса и тут же просмотреть результат в мини-браузере. Впоследствии HTML-файлы очень облегчат нам создание сайта книги в сети Интернет. Экспортированные фрагменты будут динамически вставляться в тексты страниц сайта, не требуя ручного редактирования кода страниц.

[1] Поэтому угловые скобки в именах функций не будем применять.

Кроме одной главной библиотеки, мы будем создавать несколько специальных библиотек. Главная библиотека будет загружаться всегда, а специальные — при необходимости.

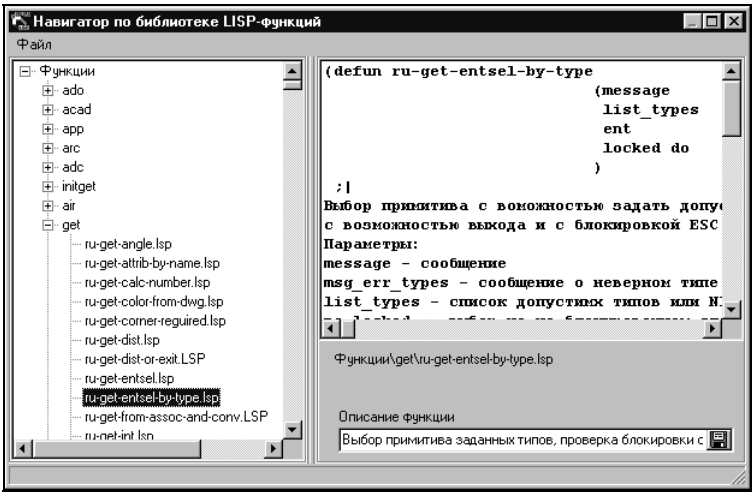


Рис. 9.2. Навигатор библиотеки функций

Файлы библиотек мы будем выделять слогом "lib", например:

- ru-lib-main.lsp (главная библиотека);
- ru-lib-window.lsp (библиотека функций рисования окон);
- ru-lib-stair.lsp (библиотека функций рисования лестниц).

Как вести дневники проекта

В начале книги мы рекомендовали постоянно документировать свою работу. Напоминаем, что для таких целей лучше всего использовать TODO-файлы. Редактирование таких файлов (дневников проекта) предусмотрено в меню **Files** (Файлы) программы Total Commander. В дневники следует постоянно записывать любые возникающие идеи, напоминания о недоделках и прочую информацию в свободной форме. При внесении конкретных изменений в исходные тексты программ мы настоятельно рекомендуем:

- не стирать, а заключать в комментарии удаляемые участки кода;
- всегда вставлять дату и инициалы того, кто вносит изменения, а также, при необходимости, дополнительный комментарий.

При внесении радикальных изменений следует в начальный комментарий к файлу внести необходимые сведения.

Как хранить данные

Большинство программ, особенно связанных с рисованием, используют какие-то постоянные (или условно постоянные данные). Организация использования таких данных оказывает существенное влияние на технологию программирования. Способ хранения данных надо продумывать заранее, по возможности оценивая перспективы использования этих же данных в других программах.

Допустим, мы разрабатываем программу DRAW_PIPE для рисования трубопроводов. Для этого нам необходимо иметь только данные стандартного номенклатурного ряда наружных диаметров. Мы можем создать простой текстовый файл со списком возможных диаметров и просто загружать его в список типа `popup_list` диалогового окна. Потом нам потребуется программа MASS_PIPE для подсчета массы трубопроводов. Для этой программы нам понадобятся данные по диаметрам и соответствующей им массе. Мы должны будем или создать новый справочник, или переделать существующий, но дополнить его сведениями о массе труб. Одновременно нам потребуется переделка программы DRAW_PIPE, а возможно, и не только ее (например, мы могли уже сделать программу LABEL_PIPE для нанесения подписей диаметров). Через некоторое время нам понадобится программа гидравлического расчета трубопроводов, для которой нужны внутренние диаметры, потом программа расчета на прочность со своими данными, и каждый раз мы будем заниматься или переделкой программ, или у нас будут плодиться файлы с различными данными по одним объектам.

Очевидно, нам необходимо или сразу разработать такую структуру данных, которая позволит решить много разных задач, или предусмотреть изменение структуры данных без переделки программ. Сделать это можно различными способами, которые мы будем рассматривать в последующих главах. В этой главе, посвященной инструментам разработчика, мы упоминаем о проблеме данных потому, что для работы с ними также нужен соответствующий инструментарий. Заботиться о нем надо заранее.

Редактор меню

Хотя нам и предстоит написать несколько сотен функций и программ, значительная часть времени будет затрачена на редактирование меню. Именно в нем пользователи увидят уже тысячи конечных пунктов. Меню системы AutoCAD мы будем редактировать традиционно, в любимом текстовом редакторе, а вот для редактирования XML-меню *(см. главу 16)* нам придется сделать визуальный редактор (рис. 9.3).

С помощью этого инструмента мы можем создавать новые и редактировать существующие XML-документы. Эта программа будет доступна только пользователям с правами на редактирование XML. Как сделана эта программа, мы описывать не будем — мало места, да и специфики AutoCAD в ней почти нет[1].

[1] О том, как сделана программа работы с XML-меню для обычных пользователей, *см. главу 16.*

Дополнительные инструменты

Каждый программист должен иметь набор дополнительных инструментальных средств. Мы уже упоминали программы для разработки справочной системы, редактирования пиктограмм, исследования и извлечения ресурсов. Помимо встроенного редактора Visual LISP потребуется мощный текстовый редактор для самых разных

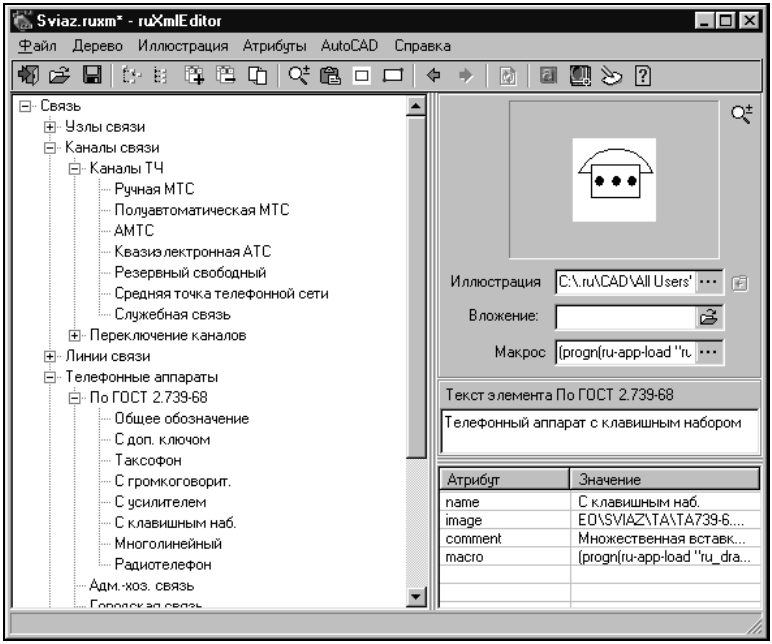


Рис. 9.3. Редактор XML-меню

работ. Мы предпочитаем применять текстовый редактор UltraEdit[1], имеющий массу дополнительных возможностей.

Особо следует остановиться на программах поиска и замены текстов. Для того чтобы реализовать соглашения по именованию файлов, функций и переменных, нам потребовалось выполнить очень большую работу по изменению имен в существующих файлах. Многофайловый поиск и замену выполняют многие программы, в том числе UltraEdit, но для каждого поиска условие необходимо задавать в диалоговом окне. Если таких условий сотни, то такая работа удовольствия не доставит. Пришлось написать собственную программу поиска и замены. Программа специально сделана без графического интерфейса, работает в режиме командной строки и очень быстро. Реальная работа по замене в 500 файлах около 600 строк (а проверить нужно в каждом файле каждую строку) выполнялась за 40 секунд на хорошем компьютере и за 4 минуты на древней машине с процессором Pentium 100.

[1] **www.ultraedit.com**.

Краткое описание этой программы мы приводим не ради рекламы, а для того, чтобы показать, как практически реализовать предложенную систему именования функций и переменных при "наличии присутствия" множества старых программ и "наличии отсутствия" в них хоть какого-то порядка.

Запуск программы производится из командной строки:

```
ru_replacer [-c] files replaces [string_symbol divider_symbol logfile]
```

где:

- `-c` — ключ, указывающий на необходимость учета регистра символов в заменяемых строках (по умолчанию регистр не учитывается);
- `files` — имя файла со списком или шаблоном имен файлов, подлежащих обработке;
- `replaces` — имя файла со списком пар искомых и заменяющих строк;
- `string_symbol` — символ, ограничивающий строки (сам замене не подлежит, по умолчанию это двойные кавычки);
- `divider_symbol` — символ-разделитель искомой и заменяющей строк в файле замен (по умолчанию это символ равно "=");
- `logfile` — имя файла протокола (по умолчанию replace.log).

Примеры запуска:

```
replacer files.lst replace.txt
replacer -c c:\temp\files.lst c:\.ru\CAD\source\lisp\replace.txt |
replacer c:\temp\files.lst c:\temp\replace.txt | =
replacer -c files.lst replace.txt | = c:\temp\report.log
```

Листинг 9.4. Пример списка файлов для замены строк files.lst

```
*.lsp
c:\.ru\cad\All Users\xml\menu\options\Ventos.xml
c:\.ru\cad\All Users\xml\menu\options\Ventrad.xml
```

Листинг 9.5. Пример файла замен replace.txt

```
ic_begin=ru-begin
ic_closeline=ru-pline-draw-closed-lw-msg
 . . .
ic_end=ru-app-end
ic_hatchreangle=ru-hatch-change-angle
ic_hatchrescale=ru-hatch-change-scale
```

Программа для каждого файла из списка проверяет каждую пару из списка замен и, при совпадении условий, производит замену. Результаты замен записываются в файл протокола. Концевые пробелы и табуляции не учитываются. Если заменяющая строка отсутствует (после разделителя нет ничего), то искомая строка удаляется. Если в заменяемой и заменяющей строке могут быть начальные и концевые пробелы, тогда должен быть задан символ-ограничитель, отличный от разделителя. В этом случае замены могут быть описаны так:

```
"00000026"="ru_draw_title_block "
```

```
"00000027"="ru_topo_title_block "
"00000028"="ru_draw_table_user "
```

Листинг 9.6. Пример файла протокола замен replace.log

```
File files.lst
393 files for replace operation was found.
File repace.txt
625 entreis for replace was found.
File ru_ar_carpet.lsp
[ic_begin] -> [ru-begin] : 1 repl.
[ic_closeline] -> [ru-pline-draw-closed-lw-msg] : 5 repl.
 . . .
[ic_end] -> [ru-end] : 1 repl.
[ic_hatchreangle] -> [ru-hatch-change-angle] : 3 repl.
[ic_hatchrescale] -> [ru-hatch-change-scale] : 4 repl.
9993 replacements was done.
```

И, в заключение главы, о самом необходимом инструменте современного программиста. Догадались, о чем речь? Конечно же, это браузер! Без доступа к Интернету, где можно найти все (кроме того, что нужно), трудно представить работу разработчика САПР. Ускорение работы не выражается в примитивном "скачивании" программ. Огромный эффект может дать и конкретный совет, и просто новый взгляд на вещи, появившийся после общения с коллегами.

Перечень наиболее интересных ресурсов Интернета мы приведем в конце книги.

ГЛАВА 10

Приемы программирования на Visual LISP

Многие специалисты, давно разрабатывающие программы на AutoLISP, не спешат переходить на Visual LISP. Разумеется, они пользуются IDE Visual LISP, но окончательный переход оттягивают. Причина обычно в том, что у них имеется много работающих программ, все, что надо, уже сделано, новые возможности изучены слабо, да они и не кажутся жизненно необходимыми. Давайте попробуем разобраться, так ли это.

Как окончательно перейти на Visual LISP

Совет

Вообще-то опытному программисту лучше всего быстро изучить все новинки, появившиеся за последние годы, путем анализа знаменитого учебного приложения Garden Path (садовая дорожка). Этот учебный пример включался во все версии AutoCAD, сейчас полностью переработан и расширен и демонстрирует все приемы работы, включая использование реакторов. В русской версии AutoCAD это учебное пособие имеет неплохо переведенную справку (файл acad_vlt.chm).

Действительно, программы, написанные на AutoLISP пятнадцать лет назад, как правило[1], будут работать и сейчас. Если они вообще работали *тогда*. Конечно, только ради перехода на модные технологии переделывать программы не нужно. Другое дело, что старые программы могут быть написаны с ошибками, о которых вы раньше не знали, и если их надо корректировать, то лучше, как минимум, знать относительно новые приемы работы.

Примечание

Не следует рассматривать эту главу, как исчерпывающий обзор возможностей Visual LISP и приемов работы с ним. Мы остановимся только на ключевых моментах. В оче-

[1] Редкое исключение составляют программы, написанные с использованием динамической модификации исходного текста функций, в том числе с самомодификацией или с применением команд, интерфейс которых изменился (INSERT, LAYER), или они оказались исключены из современных версий (AXIS, END). В таких случаях минимальная адаптация обязательна, к счастью сводится она всего лишь к замене определения функций с `defun` на `defun-q` или к корректировке нескольких строк исходного кода.

редной раз напоминаем, что эта книга не учебник и не справочник. Мы рассматриваем, как делается конкретная система и все, что не потребуется в этой системе, мы разбирать не будем.

Обзор стилей программирования

У LISP-программистов, работающих с системой AutoCAD, сложилось несколько стилей программирования. Один из них условно можно назвать "инженерным"[1], потому что так разрабатывают программы именно инженеры, вынужденные заниматься программированием (об этом мы писали во введении к книге).

"Инженерный" стиль

Этот стиль программирования, прежде всего, отличается почти исключительным использованием функции `command` для создания и редактирования объектов. Об огромной роли функции `command` в популяризации программирования на базе AutoCAD мы уже писали. До версии AutoCAD R11 создавать объекты можно было только с помощью функции `command`, это было просто и наглядно. Некоторое неудобство было в том, что существовали версии AutoCAD на разных языках с локализованными командами, и приходилось писать разные версии программ. Кроме того, в ранних версиях системы AutoCAD наблюдалось некоторое несоответствие опций команд. После появления локализованных и оригинальных имен команд *(см. главу 4)* ситуация стабилизировалась.

При использовании функции `command` разработчик должен точно знать имена команд, опции команд и возможные их варианты, т. к. он должен передать в функцию абсолютно точную последовательность ответов на возможные варианты запросов команды. Изучить эти детали довольно просто как в процессе обычной работы, так и в процессе моделирования будущей программы вызовом команд из командной строки.

Например, создать замкнутую полилинию с известными координатами вершин (точки `pt1`, `pt2`, `pt3`, `pt4`) можно следующим образом (листинг 10.1).

Листинг 10.1. Создание полилинии функцией `command`

```
(setq pt1 (list 0.0 0.0) pt2  (list 100.0 0.0) pt3 (list 100.0 50.0)
 pt4 (list 0.0 50.0))
(command "_.PLINE" pt1 pt2 pt3 pt4 "_C")
```

Нам надо было знать имя команды и как ее написать (`_.PLINE`, а не `ПЛИНИЯ`), знать, что она требует последовательного ввода точек, знать, что для замыкания полилинии требуется ввести опцию **Close** (Замкнуть), но записать ее в форме `_C`. Но, обратите внимание на то, что нас абсолютно не волновало, каким образом AutoCAD создает примитивы, в каком виде хранит их описания, и множество других "мелочей".

[1] А можно назвать и "командным", так даже солиднее.

В этом примере полилиния создается на текущем слое с текущим цветом и другими свойствами. Для изменения этих условий необходимо было бы вызвать до рисования соответствующую команду. Если вывести список свойств полученной полилинии, то мы увидим (в предположении, что текущее значение системной переменной PLINETYPE заставляет систему формировать компактные (LWPOLYLINE), а не подробные (POLYLINE) полилинии):

```
((-1 . <Entity name: 40074e70>) (0 . "LWPOLYLINE")
(330 . <Entity name: 40074cf8>) (5 . "76") (100 . "AcDbEntity") (67 . 0)
(410 . "Model") (8 . "0") (100 . "AcDbPolyline") (90 . 4) (70 . 129)
(43 . 0.0) (38 . 0.0) (39 . 0.0) (10 0.0 0.0) (40 . 0.0) (41 . 0.0)
(42 . 0.0) (10 100.0 0.0) (40 . 0.0) (41 . 0.0) (42 . 0.0)
(10 100.0 50.0) (40 . 0.0) (41 . 0.0) (42 . 0.0) (10 0.0 50.0) (40 . 0.0)
(41 . 0.0) (42 . 0.0) (210 0.0 0.0 1.0))
```

И все это мы создали посредством короткой строки! Впоследствии придется разбираться и с этими списками, изучать DXF-коды, но первый результат получен легко.

После появления методов создания объектов с помощью функции `entmake` и изменения их свойств с помощью функции `entmod` сформировался стиль программирования, который условно можно назвать "программистским". Такой стиль стали применять профессиональные программисты, которых *вынудили* заниматься разработками для AutoCAD, и инженеры (особенно достигшие пределов возможностей функции `command`), которых *жизнь заставила* стать программистами.

Особенности функции *command*

Хотя функция `command` используется с незапамятных времен, мы напомним некоторые особенности ее применения. Аргументами функции `command` являются имена команд, их опции и данные, которые ожидает команда.

Имена команд

В очередной раз повторяем:

Имена команд должны начинаться с префикса "_.", опций — с префикса "_".

Не обращайте внимания, что во многих "фирменных" примерах префиксы не используются. Вспомните любимую фразу Михаила Задорнова!

Точка в префиксе команды означает, что используется "настоящая" команда, а не переопределенная пользователем. Переопределение команд встречается, хоть и редко. Приведем пример. Загрузим функцию:

```
(defun C:LINE ()
  (princ"Это переопределенная команда")
 (princ)
)
```

А теперь поэкспериментируем:

```
Command: undefine
Enter command name: line
```

Пытаемся выполнить команду LINE (ОТРЕЗОК):

```
Command: line
Это переопределенная команда
Command:
Command: _line
Это переопределенная команда
```

Вводим команду с точкой:

```
Command: .line
Specify first point:
Specify next point or [Undo]:
```

Заработало! Восстанавливаем команду LINE (ОТРЕЗОК) в прежнем виде:

```
Command: redefine
Enter command name: line
Command: line
Specify first point:
```

Конечно, при переопределении команды стараются просто добавлять к существующей команде дополнительные функции. Например, мы когда-то добавляли к команде LINE (ОТРЕЗОК) сохранение файла через заданный промежуток времени (автосохранения тогда не было). Иногда пытаются переопределить команды записи файла, делая демонстрационную версию и прочие милые глупости.

Команды с диалоговыми окнами обычно можно вызывать с префиксом "-". Это сделано специально для того, чтобы их можно было использовать внутри функции `command`.

Использование GET-функций

Использовать функции ввода `getxxx` внутри `command` было нельзя. Но очень хотелось, и стало можно. В рассматриваемых нами версиях системы AutoCAD можно применить такую конструкцию:

```
(command "_.LINE" (setq pt1 (getpoint "\nLINE2- От точки: "))
(getpoint pt1 "\nДо точки: ") "")
```

и все сработает. Однако *можно* применять не означает, что так и *нужно* делать. В таком случае нет гарантии надежной работы (пользователь может не указать точку, нажать клавишу и т. д.). Лучше все-таки пользоваться надежными методами вычисления требуемых данных до вызова функции `command`.

Аргументы, передаваемые команде, могут быть строками, числами и точками, т. е. такими, какие ожидает получить выполняемая команда в ответ на ее запросы и именно в том порядке, в каком эти запросы следуют. Число (константу) можно передать и в виде строки, и в виде числа. Передача числа в виде строки иногда даже предпочтительна, например:

```
(command "_.INSERT" block_name insert_point 1 1 (angtos angle_rotate 1 4))
```

В этом примере масштабы вставки блока переданы в виде чисел, а угол поворота в виде строки, потому, что при вставке блока команда ожидает ввода угла в градусах.

Пустая строка равносильна нажатию клавиши <Enter>, а вызов (command) без аргументов равносилен нажатию клавиши <Esc>. Примеров вы насмотритесь в следующих главах.

Символ pause в строке аргументов приостанавливает действие функции для ввода пользователем данных.

Внимание!

При проверке текста программы в IDE Visual LISP обычно копируют обнаруженный список глобальных переменных и вставляют его в определение функции для объявления этих переменных локальными. Символ pause непременно попадет в этот список, а, объявив его локальной переменной, вы получите загадочную неработоспособность функций.

Прерывание функции *command*

Иногда требуется искусственно прервать незавершенную команду. Для этого используется вызов (command) без аргументов. Например, при внедрении в рисунок библиотеки блоков из внешнего файла нам необходимо, чтобы в рисунке появилось определение блока (и всех вложенных в него блоков), но не было самой вставки блока. Это можно сделать так:

```
(command "_.INSERT" file_name)(command)
```

В этом примере мы имитируем нажатие клавиши <Esc> в тот момент, когда система AutoCAD уже внедрила описание блока в рисунок и выдала запрос на точку вставки. Отменяя команду, мы отменили вставку блока, но сохранили его определение (блок с именем файла и все вложенные блоки остались в таблице BLOCK).

Незавершенная команда

Встречаются ситуации, в которых неизвестно, сколько запросов будет выдавать команда и какие могут быть ответы на эти запросы. Вот несколько надуманный, но наглядный пример:

```
(command "_.PLINE")
(while (= (logand (getvar "CMDACTIVE") 1) 1)
(command pause)
)
```

Мы вызвали команду рисования полилинии и завершили функцию command, закрыв скобку. Но команда при этом осталась активной и вывела свое первое стандартное приглашение. Далее мы в цикле, пока команда активна, вызываем функцию command с параметром pause, т. е. даем возможность пользователю выполнить любой допустимый для этой команды ввод (указывать точки, изменять ширину и т. п.).

Еще один пример. При установке стиля текста мы в цикле имитируем неизвестное заранее количество требуемых ответов в виде нажатия клавиши <Enter>. Обратите внимание на то, с каким префиксом мы вызываем имя команды:

```
(command "_.-STYLE" style_name
 (if (findfile shx_file_name) shx_file_name (getvar "FONTALT"))
 text_height text_width text_oblique
 (while (= 1 (logand 1 (getvar "CMDACTIVE"))) "")
)
```

Особенности функции *vl-cmdf*

Функция command обладает рядом недостатков, для устранения которых в Visual LISP была введена новая функция

```
(vl-cmdf [аргументы])
```

В отличие от функции command, начинающей выполнять команду сразу, vl-cmdf вычисляет все аргументы *до начала* выполнения команды и, если в аргументах обнаруживаются ошибки, команда не выполняется. Функция command всегда возвращает nil, функция vl-cmdf — T при успешном выполнении и nil при обнаружении ошибки.

Пример:

```
(vl-cmdf "_.LINE" (setq pt1 (getpoint "\nLINE2- От точки: "))
(getpoint pt1 "\nДо точки: ") "")
```

Вот еще один пример. Развивать его мы будем при разработке реальной функции многократной вставки блока. Пока мы просто пытаемся многократно вставить блок с именем КРУГ, не имеющий атрибутов.

Вариант с функцией command:

```
Command: (while (command "_.INSERT" "КРУГ" pause 1 1 1))
nil
```

Эта конструкция прерывается после первой же вставки блока. Вариант с функцией vl-cmdf:

```
Command: (while (vl-cmdf "_.INSERT" "КРУГ" pause 1 1 1))
```

Мы можем много раз вставлять блок, указывая точку вставки. Изображение блока висит на курсоре. При нажатии клавиши <Enter> появляется сообщение:

```
Point or option keyword required.
Specify insertion point or [Scale/X/Y/Z/Rotate/PScale/PX/PY/PZ/PRotate]:
```

Прервать вставку можно только нажатием клавиши <Esc>:

```
*Cancel*
nil
```

Попробуйте самостоятельно найти причины разной работы этих вариантов.

Команда VBASTMT

Эта команда имеет особое значение. Она запрашивает выражения, которые могут выполняться в VBA. При вызове из командной строки особой пользы от этого нет, но если вызвать эту команду через функцию command, то можно передать в виде строки любое синтаксически правильное выражение VBA. Так как передана будет одна строка, то операторы нужно разделять двоеточиями так, как это нужно делать при размещении их в одной строке в программе на VBA. Такой прием удобен, когда средства VBA *действительно* упрощают решение задачи.

Например, сделать активным любой из открытых документов можно так:

```
(vl-cmdf "_.VBASTMT"
(strcat "ThisDrawing.Application.Documents.Item(0).Activate"))
```

В этом примере мы активизировали документ, имеющий номер 0 в семействе открытых документов. При этом мы, забежав вперед, обратились к объектной модели AutoCAD, да еще и средствами VBA. Но в данном случае такое смешение стилей оправдано.

"Программистский" стиль

Такой стиль характеризуется тем, что при работе с функцией `entmake` разработчик собирает в ассоциативный список информацию о координатах, слое и цвете объекта, а затем передает все это в AutoCAD для построения. Изменение свойств объекта осуществляется функцией `entmod` с передачей ей аналогичного списка. Ассоциированный список функции `entmake` очень похож на список, получаемый функцией `entget`. Работа со списками полностью соответствует идеологии языка LISP, придает программам большую гибкость и выигрыш в скорости работы. Но разработчик должен знать уже гораздо больше о структуре данных, прежде всего DXF-коды различных объектов, и должен уметь свободно оперировать со списками.

Создать замкнутую полилинию с известными координатами вершин (точки `pt1`, `pt2`, `pt3`, `pt4`) можно следующим образом (листинг 10.2).

Листинг 10.2. Создание полилинии функцией `entmake`

```
(setq pt1 (list 0.0 0.0) pt2 (list -100.0 0.0) pt3 (list -100.0 -50.0)
 pt4 (list 0.0 -50.0))
(entmake
    (list ;; формируем ассоциированный список
          '(0 . "LWPOLYLINE")      ;; Тип примитива
          '(100 . "AcDbEntity")    ;; Маркер подкласса
          '(8 . "TEST")            ;; Слой
          '(100 . "AcDbPolyline") ;; Маркер подкласса
          '(90 . 4)                ;; Количество вершин
          '(70 . 1)                ;; Флаг замкнутой полилинии
          (cons 10 pt1)           ;; Точка вершины 1
          (cons 10 pt2)           ;; Точка вершины 2
          (cons 10 pt3)           ;; Точка вершины 3
          (cons 10 pt4)           ;; Точка вершины 4
   )
  )
```

В этом примере мы сформировали ассоциированный список, через который передаем функции `entmake` свойства создаваемой полилинии. Для формирования списка нам нужно было знать его структуру, DXF-коды и правила формирования списка. Это сложнее, чем вариант с `command`, но дает больше возможностей. Например, мы указали слой, на котором должна создаваться полилиния, этот слой был создан, но не стал текущим. Мы могли бы задать еще много других свойств, но должны были бы знать их DXF-коды.

Если вывести список свойств полученной полилинии, мы увидим:

```
((-1 . <Entity name: 40074e98>) (0 . "LWPOLYLINE")
(330 . <Entity name: 40074cf8>) (5 . "7B") (100 . "AcDbEntity") (67 . 0)
(410 . "Model") (8 . "TEST") (100 . "AcDbPolyline") (90 . 4) (70 . 1)
```

```
(43 . 0.0) (38 . 0.0) (39 . 0.0) (10 0.0 0.0) (40 . 0.0) (41 . 0.0)
(42 . 0.0) (10 -100.0 0.0) (40 . 0.0) (41 . 0.0) (42 . 0.0)
(10 -100.0 -50.0) (40 . 0.0) (41 . 0.0) (42 . 0.0) (10 0.0 -50.0)
(40 . 0.0) (41 . 0.0) (42 . 0.0) (210 0.0 0.0 1.0))
```

Результат такой же, как и при использовании функции `command`. Отличаются имена слоев и координаты вершин, которые мы и задавали другие.

"Объектный" стиль

После внедрения в AutoCAD технологии ActiveX начал формироваться "объектный" стиль программирования. Объектный стиль характеризуется тем, что разработчик работает не с именами и опциями команд и не с ассоциированными списками свойств различных объектов, а с объектной моделью AutoCAD через функции ActiveX. Создание объектов в Visual LISP с помощью функций ActiveX является самым современным из имеющихся методов. Это действительно так, если под *современным* понимать новый или "молодой". Считается, что технология ActiveX обладает рядом преимуществ по сравнению с использованием функций `entmake` и `command`, потому что:

- функции ActiveX работают быстрее, чем функция `command`;
- имена функций ActiveX обозначают действия, которые они выполняют, что обеспечивает более удобное чтение, обновление и исправление программ.

Давайте разберемся, что здесь правда, а что "рекламная шелуха". Со скоростью разберемся попозже, пока проанализируем удобства работы. Попробуем создать нашу полилинию объектными методами. Пусть мы снова зададим координаты четырех точек.

```
(setq pt1 (list 0.0 0.0) pt2 (list -100.0 0.0) pt3 (list -100.0 -50.0)
 pt4 (list 0.0 -50.0))
```

"Полазав по дереву" объектной модели AutoCAD, мы найдем метод создания компактных полилиний:

```
(vla-addlightweightpolyline ObjSpace PointsList)
```

Выясним, что `ObjSpace` — это VLA-объект пространства (листа или модели), в которое надо добавить полилинию, а `PointsList` — *вроде бы* список координат. Требуемое пространство мы должны передать в виде именно объекта, а не какого-то символьного обозначения. Допустим, нас интересует пространство модели. Для того чтобы получить *объект пространства модели*, мы должны обратиться к *объекту текущего документа*, а документ запросить у самого AutoCAD, но тоже как у *объекта приложения*. Выглядеть это может так:

```
(setq acad_app     (vlax-get-acad-object)
      acad_doc     (vla-get-ActiveDocument acad_app)
      model_space  (vla-get-ModelSpace acad_doc)
)
```

Теперь, если мы создадим список точек

```
(setq pnt_lst (list pt1 pt2 pt3 pt4)),
```

а затем попытаемся выполнить

```
(vla-AddLightweightPolyline model_space pnt_lst),
```

то получим сообщение об ошибке:

```
; error: lisp value has no coercion to VARIANT with this type: ((0.0 0.0)
(-100.0 0.0) (-100.0 -50.0) (0.0 -50.0))
```

Система AutoCAD явно возмущается тем, что вместо типа VARIANT мы подсовываем ей данные такого типа, который она даже не может назвать по имени. Мы-то знаем, что (type pnt_lst) вернет LIST, что же, система AutoCAD об этом забыла, она ведь только что сама создала этот список? Здесь нас подводит "лисповское" мышление. Обращаемся-то мы к объектной модели AutoCAD, реализованной в виде COM-сервера, написанного на C++, а там нет данных типа LIST, но есть тип VARIANT, для которого, в данном контексте, требуется передавать данные в виде безопасного массива (SAFEARRAY).

Мы составили список точек по логике LISP — каждая точка представлена списком координат X и Y, а весь список является списком списков координат точек. Не хватает еще только флага "замкнутости", но о нем мы пока помалкиваем.

По "объектной" же логике, список точек должен быть представлен простой последовательностью чисел, в нашем случае четырех двухмерных точек это будет x1 y1 x2 y2 x3 y3 x4 y4. На LISP это можно написать сначала в виде списка:

```
(setq pnt_lst (list 0.0 0.0 -100.0 0.0 -100.0 -50.0 0.0 -50.0)),
```

а затем произвести ряд манипуляций с данными:

```
(setq safe_array_pnt_lst (vlax-make-safearray vlax-vbdouble '(0 . 7)))
```

Сейчас мы создали безопасный массив из восьми (четыре точки по две координаты на точку, нумерация с нуля) вещественных чисел двойной точности. Фактически мы выделили память для размещения массива чисел и получили в переменной safe_array_pnt_lst указатель на эту область памяти. Теперь мы должны заполнить этот пока пустой массив значениями.

```
(setq pnt_lst_len (length pnt_lst))
(setq i 0)
(while (< i pnt_lst_len)
  (vlax-safearray-put-element safe_array_pnt_lst i (nth i pnt_lst))
  (setq i (1+ i))
)
```

Это еще не все. Теперь требуется создать тип данных VARIANT из безопасного массива.

```
(setq var_pnt_lst (vlax-make-variant safe_array_pnt_lst
                    (logior vlax-vbarray vlax-vbdouble)
                  )
)
```

Только теперь мы можем создать объект полилинии, указать, что полилиния должна быть замкнута, и, для примера, установить цвет 1:

```
(setq pl_obj (vla-addlightweightpolyline model_space var_pnt_lst))
(vla-put-closed pl_obj :vlax-true)
(vla-put-color pl_obj 1)
(vla-update pl_obj)
```

Как видите, нам пришлось проделать немало манипуляций с данными. Использование объектных методов далеко не так просто, как в этом уверяют. Да, нам не нужно

знать DXF-кодов, но следует знать объектную модель, нужно заботиться о выделении памяти и о преобразовании типов. Мы ведь в этом примере все упростили, на самом деле нужны еще и дополнительные проверки и преобразования. Например, мы создавали список вершин из заведомо двухмерных точек, а если бы он возвращался какой-то функцией, с трехмерными координатами? Произошла бы ошибка.

Так что же, объектный метод действительно так плох и неудобен? Ни в коем случае! "Плох" он с точки зрения традиционного LISP-программиста. Слишком многое, с его точки зрения, делается через это самое место. Но, представьте, что разработчики системы AutoCAD создали бы ее объектную модель, ориентируясь на LISP. Кому она была бы нужна? Уж точно не LISP-программистам, они и раньше имели почти все, что им требовалось. Но с AutoCAD тогда не смог бы работать никто другой, мы вернулись бы примерно к той ситуации, которая была в версии AutoCAD R13. Другое дело, если бы компьютерные технологии развивались иным путем (мы намекали на это в *главе 9*), операционные системы писались бы на LISP, не было бы этих "ужасных" C++, Pascal, Basic. "То-то было б весело, то-то хорошо". Возможно, в одном из параллельных миров так и есть, но мы-то живем здесь и сейчас. В нашем компьютерном мире "правит бал" известная фирма, мы работаем по технологиям, которые она диктует, и должны, пусть с ворчанием, подчиняться ее законам. Спасибо хоть фирме Autodesk, что она ввела дополнительную технологию, оставив и старые. Пока оставив.

Избавиться от многих сложностей мы можем просто. Для того чтобы не мучиться с рисованием различных объектов, мы будем просто разрабатывать высокоуровневые функции, внутри которых будут скрыты детали реализации. Например, для рассмотренного случая с полилинией у нас может быть разработана функция

```
(ru-pline-add <список_точек> <замкнутость> <ширина> <слой> <цвет>)
```

Такой функции можно передавать обычный список точек, а факультативные параметры (ширина, слой, цвет) при рисовании с текущими установками заменять на `nil`. Прикладной программист не будет знать, как реализовано рисование внутри этой функции. Это будет единственное место в системе, где происходит создание объекта, и это место мы всегда сможем переписать по вкусу.

> А теперь попробуйте, в качестве "домашнего задания" создать на VBA подобную функцию, единственную на всю систему и доступную из всех модулей.

Может быть, на VBA создавать объекты легче? Давайте посмотрим.

Листинг 10.3. Создание полилинии на VBA

```
Sub Example()
Dim pl_Obj As AcadLWPolyline
Dim var_pnt_lst (0 To 7) As Double
 var_pnt_lst (0) = 0
 var_pnt_lst 1) = 0
 var_pnt_lst (2) = -100
 var_pnt_lst (3) = 0
 var_pnt_lst (4) = -100
 var_pnt_lst (5) = -50
 var_pnt_lst (6) = 0
 var_pnt_lst (7) = -50
```

```
Set pl_Obj = ThisDrawing.ModelSpace.AddLightWeightPolyline(var_pnt_lst)
End Sub
```

На вид, кажется, действительно проще. Преобразований типов нет, потому что сразу создается массив требуемого типа. Трудно понять, где кончается одна точка, а где начинается другая, так на это можно наплевать. Все создание объекта записано в одну строку, потому что есть предопределенный объект `ThisDrawing`, а далее вызываются его объекты. Для справедливости приведем и аналогичный LISP-вариант с оптимизированным кодом (листинг 10.4).

Листинг 10.4. Создание полилинии объектными методами

```
(defun example ()
(vla-put-closed
 (vla-addlightweightpolyline
  (vla-get-ModelSpace
   (vla-get-ActiveDocument (vlax-get-acad-object))
 )
  (vlax-make-variant
   (vlax-safearray-fill (vlax-make-safearray vlax-vbdouble '(0 . 7))
     (list 0.0 0.0 100.0 0.0 100.0 -50.0 0.0 -50.0)
   )
   (logior vlax-vbarray vlax-vbdouble)
  )
) :vlax-true)
)
```

Ощутили разницу? У нас нет ни одной переменной, все сделано "по-лисповски", значения одной функции передаются на вход другой. И строк столько же, хотя разбили мы текст на строки только для улучшения читабельности. И, если вынести константы в параметры, эта функция может стать универсальной, но не очень надежной. О надежности поговорим позже.

Сравнение скорости работы

Теперь разберемся со скоростью работы. Загрузите примеры, поставляемые с AutoCAD (%ACAD% обозначает корневой каталог, в котором установлен AutoCAD на вашем компьютере):

- %ACAD%\Sample\VisualLISP\activex\al-tst.lsp
- %ACAD%\Sample\VisualLISP\activex\vla-tst.lsp

и выполните команды AL-TST и VLA-TST. Эти программы выполняют одинаковую работу (рисуют 2 000 полилиний, изменяют их цвет и удаляют), но разными методами, с фиксацией затраченного времени. AL-TST использует функции `command` и `entmod`, а VLA-TST — функции ActiveX. Результаты (конечно, разные на различных компьютерах) показывают, что функции ActiveX работают значительно быстрее. На очень "слабом" компьютере, специально используемом нами для тестирования, получены результаты:

- AL-TST — 206.8 сек. (13.141 сек. на "хорошем" компьютере)
- VLA-TST — 143.79 сек. (5.61 сек. на "хорошем" компьютере)

При замене в al.lsp `(command "_.PLINE"...)` на `(entmake...)` получаем результаты:

- AL-TST — 115.95 сек. (6.812 сек. на "хорошем" компьютере)

Естественно, что вариант с функцией `command` показывает худшие результаты. Иного и не могло быть, т. к. AutoCAD должен анализировать (неизвестный заранее) поток данных, передаваемый в виде аргументов функции. При использовании функции `entmake` передаются уже известные структуры данных, близкие к внутреннему представлению, и процесс идет гораздо быстрее. При использовании объектной модели данные передаются в виде уже совсем близком к внутреннему представлению (через Visual LISP) и в "готовом к употреблению" виде при использовании VBA. Некоторое замедление дает цепочка посредников при передаче данных через интерфейсы различных объектов. Именно это дает действительное или кажущееся отставание. На высокопроизводительных машинах накладные расходы будут менее заметны. В целом можно подтвердить, что использование ActiveX дает выигрыш в производительности.

Теперь рассмотрим вопрос с другой стороны. А *что* нам приходится рисовать в своих программах? Пример с 2 000 полилиниями выглядит надуманным. Чаще всего рисуется небольшое количество объектов, а большую часть времени AutoCAD просто ожидает ввода пользователя. Если вам нужно нарисовать не две тысячи, а один объект, то не имеет значения, нарисуется ли он за 0.001 секунды или за 0.003 секунды, разницу вы не ощутите.

Примечание

Однако в трехмерной модели корпуса судна за один расчет приходится строить сразу десятки и сотни полилиний, представляющих собой границы больших и малых конструкций (см. **www.private.peterlink.ru/poleshchuk/cad/steel2.html**). *(Николай Полещук)*

Почему бы окончательно не избавиться от *command*

С помощью функции `entmake` или методами ActiveX объект создается за один прием. Координаты и другие характеристики объекта вычисляются предварительно. В реальной работе это не всегда возможно. Пожалуй, даже чаще требуется интерактивная работа. Мы в дальнейшем будем разрабатывать много программ, требующих интерактивного ввода данных. Типичный пример — трассы различных коммуникаций. Обычно трассу нужно провести по местности, не зная координат, а выбирая направление вручную. Выполнить это можно только с помощью функции `command`. Другой пример — вставка блоков. Точку вставки блока нужно задавать, имея *висящее* на курсоре изображение блока (предварительно масштабированное и повернутое). Таких примеров имеется множество. Так что, если бы функции `command` вообще бы не было, ее надо было бы немедленно выдумать.

Промежуточный диагноз

Ориентироваться только на один стиль программирования нельзя, особенно при разработке большой сложной системы. Современный разработчик просто обязан знать все технологии, их преимущества и недостатки, и умело применять то, что выгоднее всего в конкретной ситуации. Например, работать с наборами гораздо

удобнее с использованием традиционных методов, а работать со свойствами самого AutoCAD (семейство `Preferences`) — через объектную модель.

Если ваши программы написаны "старинным стилем", но устраивают ваших пользователей — пусть они такими и остаются. Нечего комплексовать из-за того, что они не соответствуют сегодняшней моде. Но когда вы столкнетесь с задачами другого уровня, вполне может оказаться, что решить их можно только с использованием технологий ActiveX.

Использование объектной модели AutoCAD

Объектная модель приложения является совокупностью объектов, свойств, методов и событий. Для каждого из элементов этой модели имеется своя реализация в виде данных и функций, обеспечивающих взаимодействие с пользователем.

Объекты ActiveX в AutoCAD рассматриваются как иерархия, содержащая не только примитивы, но и прочие элементы (таблицы, словари и т. д.). Однотипные объекты объединяются в семейства (Collections). Корневым объектом этой модели является объект `Application`, связанный с семействами `Preferences`, `Documents`, `MenuBar`, `MenuGroups`. Каждое из этих семейств имеет множество "потомков". В документации по VBA приведено иерархическое дерево системы AutoCAD. Ознакомиться (и не только ознакомиться, а постоянно работать) с объектной моделью можно, заглянув в файл справки acadauto.chm[1]. К сожалению, *полностью* объектную модель, да на русском языке, да и с примерами не только на VBA, кажется, еще никто не описал. В этой книге мы также не имеем возможности привести даже сокращенное описание. Мы будем рассматривать только конкретные объекты, которые нам понадобятся для написания соответствующих программ.

Следует понимать, что не все, что умеет AutoCAD, отражено в объектной модели. Технология ее создания заключается в том, что после того, как написано обычное приложение Windows, его наделяют функциями автоматизации. Для этого код дополняют соответствующими методами и объектом автоматизации[2]. Выполняют это люди, а людям свойственно ошибаться. Если интерфейс какого-то объекта не объявлен при формировании сервера автоматизации, то до этого объекта невозможно добраться ни через Visual LISP, ни через VBA. Поэтому особый интерес вызывает, *что именно* недоступно через объектную модель, но, возможно, доступно через `command` или DXF-коды.

При работе через объектную модель важно правильно использовать методы и знать, применимы ли они к данному объекту в данной ситуации. Например, при создании полилинии с помощью функции `entmake` (см. листинг 10.2) мы назначили создаваемому примитиву несуществующий слой, и такой слой был создан. Если бы мы попытались при создании полилинии объектными методами (см. листинг 10.4) применить функцию `vla-put-layer` с именем несуществующего слоя, мы получили бы ошибку.

---

[1] Объектная модель описана также в главе 43 книги "AutoCAD 2002" и в главе 1 книги "AutoCAD 2004: разработка приложений и адаптация" (обе книги издательства "БХВ-Петербург" в серии "В подлиннике").

[2] Как это делается, мы рассмотрим в следующих главах.

Пример исследования объектной модели

В своей системе мы будем использовать тысячи блоков. Мы знаем, что при создании описания блока можно добавить пояснение (description), а при использовании Центра управления системы AutoCAD мы видим, что блоки могут иметь и миниатюры (растровые образцы) для предварительного просмотра. Очевидно, что все это хранится в базе данных рисунка, и нам хотелось бы добираться программным путем хотя бы до пояснения. В диалоговом окне свойств пояснение к блоку не отображается. Получив свойства блока через `(entget (car (entsel))`, мы видим что-то наподобие следующего:

```
((-1 . <Entity name: 4009b018>) (0 . "INSERT")
(330 . <Entity name: 4008acf8>) (5 . "BB") (100 . "AcDbEntity")
(67 . 0) (410 . "Model") (8 . "0") (100 . "AcDbBlockReference")
(2 . "КРУГ") (10 3.84466 2.64426 0.0) (41 . 1.0) (42 . 1.0) (43 . 1.0)
(50 . 0.0) (70 . 0) (71 . 0) (44 . 0.0) (45 . 0.0) (210 0.0 0.0 1.0))
```

Естественно, что здесь мы не увидим свойств самого блока, это только свойства вхождения блока (примитива INSERT), из которых мы узнаем имя блока (КРУГ) и параметры точки вставки. Собственно имя блока нам и требовалось. Теперь мы ищем блок в таблице блоков

```
Command: (setq blk (tblsearch "BLOCK" "КРУГ"))
((0 . "BLOCK") (2 . "КРУГ") (70 . 0) (4 . "Просто круг") (10 0.0 0.0 0.0)
(-2 . <Entity name: 4009b010>))
```

и сразу видим подозрительный код 4 с чем-то похожим на пояснение к блоку "Просто круг". Ознакомившись с "DXF Reference"[1], устанавливаем, что пояснение к блоку действительно должно быть связано с кодом 4. Перетащив этот блок в другой рисунок через Центр управления и, уже в другом рисунке, разыскав этот блок, мы убеждаемся, что пояснение присутствует и в этом рисунке. Это нам открывает определенные перспективы.

Теперь попробуем сделать то же через объектную модель. Вначале напишем простенькую программу для просмотра свойств объекта (листинг 10.5).

Листинг 10.5. Функция `C:RU_ENT_DUMP`

```
(defun C:RU_ENT_DUMP(/ ent)
(while (setq ent (entsel
"\nВыбери объект для просмотра свойств <Выход>: \n"))
(princ "\nСвойства объекта:\n")
(vlax-dump-object(vlax-ename->vla-object (car ent)))(princ)
)
(princ)
)
```

Между делом, для сравнения технологий напишем аналогичную программу, работающую через функцию `entget` (листинг 10.6). Эта программа позволяет просматривать свойства, включая вложенные объекты (атрибуты и вершины полилиний).

---

[1] Или с главой 29 книг "AutoCAD 2002" и "AutoCAD 2004".

Листинг 10.6. Функция C:RU_ENTLST

```
(defun C:RU_ENTLST (/ ent elst ename etyp)
(while
 (setq ent (entsel "\nВыбери объект для просмотра свойств <Выход>: \n"))
  (princ "\nСвойства объекта:\n")
  (setq ename (car ent))(setq elst (entget ename))(princ "\n\n")
  (princ elst)
  (if (= (cdr (assoc 66 elst)) 1)
    (while
     (and (entnext enam)
            (or(= (setq enam (entnext ename)
                        elst (entget ename)
                        etyp (cdr (assoc 0 elst))
                  );_ end of setq
                    "ATTRIB"
                )
                 (= etyp "VERTEX")
                 (= etyp "SEQEND")
                );_ end of or
              );_ end of and
  (princ "\n\n")
  (princ elst)
);_ end of while
);_ end of if
(princ)
);_ end of while
(princ)
);_ end of defun
```

Просмотрев свойства вставки блока командой RU_ENT_DUMP, мы видим:

```
Выбери объект для просмотра свойств <Выход>:

Свойства объекта:
; IAcadBlockReference: AutoCAD Block Reference Interface
; Property values:
;   Application (RO) = #<VLA-OBJECT IAcadApplication 00a88728>
;   Color = 256
;   Document (RO) = #<VLA-OBJECT IAcadDocument 00f2f93c>
;   Handle (RO) = "80"
;   HasAttributes (RO) = 0
;   HasExtensionDictionary (RO) = 0
;   Hyperlinks (RO) = #<VLA-OBJECT IAcadHyperlinks 00fac0f4>
;   InsertionPoint = (203.086 285.858 0.0)
;   Layer = "0"
;   Linetype = "ByLayer"
;   LinetypeScale = 1.0
;   Lineweight = -1
;   Name = "КРУГ"
;   Normal = (0.0 0.0 1.0)
```

```
;   ObjectID (RO) = 1074220736
;   ObjectName (RO) = "AcDbBlockReference"
;   OwnerID (RO) = 1074220280
;   PlotStyleName = "ByLayer"
;   Rotation = 0.0
;   Visible = -1
;   XScaleFactor = 1000.0
;   YScaleFactor = 1000.0
;   ZScaleFactor = 1000.0
```

Конечно, это только свойства вставки. Мы их показали, чтобы убедиться в том, что объектная модель выдает больше свойств, чем функция `entget`.

Начинаем изучать объектную модель. Напишем функцию для получения списка имен блоков (листинг 10.7)

Листинг 10.7. Функция `ru-list-block-names`

```
(defun ru-list-block-names ( / item_name result)
(setq result '())
(vlax-for each
  (vlax-get-property
   (vla-get-activedocument
    (vlax-get-acad-object)
  ) 'blocks
 )
   (setq item_name (vlax-get-property each 'Name)
         result (cons item_name result)
  );_ end of setq
);_ end of vlax-for
(reverse result)
)
```

Это временный вариант функции. Впоследствии мы приспособим его для вывода списка членов любого семейства. Вызов этой функции дает такой результат:

```
Command: (ru-list-block-names)
("*Model_Space" "*Paper_Space" "*Paper_Space0" "_Dot" "КРУГ")
```

Примечание

Кстати, такой же результат мы могли получить и безо всякой объектной модели:

```
(ai_table "BLOCK" 0)
("КРУГ" "_Dot")
```

Здесь отсутствуют `"*Model_Space"`, `"*Paper_Space"` и `"*Paper_Space0"`, но это как раз и ненастоящие блоки.

Попробуем все-таки найти свойство `Description` для блока. Поиск в справочной системе результатов не дает. Свойство `Description` упоминается только для определения типа линии. Блуждание по объектам `Block`, `BlockRef`, семейству `Blocks` и другим, имеющим хоть какое-то отношение к блокам, результатов не дает.

Мы не поленились поискать в Интернете. На всех сайтах, включая китайские, `Description` для блоков упоминается только в связи с вводом в диалоговом окне создания блока. Поиск в Autodesk Knowledge Base выявил пример извлечения пояснения к блоку через DXF-код. Это мы и сами умеем. Может быть, мы были не очень настойчивы, а вы, наши читатели, все-таки нашли это свойство в объектной модели? Во всяком случае все оказалось не так легко и просто, как уверяет фирма Autodesk. Возможно, это именно тот случай, не опубликованный в объектной модели. Именно то, что нам нужно, и отсутствует. Как всегда (закон бутерброда).

Несмотря на эту неудачу, объектную модель мы будем использовать широко. Иногда без нее не обойтись. Мы уже упоминали *(см. главу 8)* о единственном способе массового управления доступом к пунктам меню. Приведем еще один интересный пример. Мы имеем некую программу (не будем ее называть), которая создает в AutoCAD необходимый нам рисунок, но не позволяет его сохранить в виде файла. Программой блокируются все команды, позволяющие сохранить файл. А через объектную модель это делается просто:

```
(vla-SaveAs
 (vla-get-activedocument (vlax-get-acad-object))
  "C:\\TEMP\\1.dwg" acR14_DWG)
```

Как использовать объектную модель

Теперь разберем несколько основных правил, которыми необходимо руководствоваться при работе с объектной моделью AutoCAD. Прежде всего, развеем суеверия по поводу функции `vl-load-com`. Вызов этой функции делают в каждом примере работы с AutoCAD через ActiveX, и некоторые программисты на этом основании считают, что это "загрузка Visual LISP". Ничего подобного! В AutoCAD 2000 (R15) и старше Visual LISP поставляется с системой и его библиотеки загружаются автоматически вместе с AutoCAD. Функция `vl-load-com` сама является функцией Visual LISP, загружающей дополнительную DLL-библиотеку, обеспечивающую дополнительные возможности работы с объектной моделью. Так как эти когда-то дополнительные возможности стали уже основными, то в AutoCAD 2004 библиотека для работы с объектной моделью загружается автоматически. Лишний вызов `(vl-load-com)` не вредит, но мы его будем делать только один раз.

Некоторое смущение у AutoLISP-программистов вызывают непривычные типы данных при использовании объектной модели.

Вызвав `(vlax-get-acad-object)`, мы получим:

```
#<VLA-OBJECT IAcadApplication 00a88728>
```

Это непонятно, но ничем не лучше результата, полученного через `(entlast)`:

```
<Entity name: 4008c220>
```

В первом случае мы получили указатель на VLA-объект, во втором случае — на имя примитива. Чтобы получить "человеческие" данные, нам в любом случае необходимо проделать дополнительные действия. В функции `ru-list-block-names` (см. листинг 10.7) мы последовательно получали несколько указателей: сначала указатель на объект `AcadApplication`, в этом объекте — указатель на `ActiveDocument`, в документе — указатель на семейство блоков и только затем, проходя по каждому элементу этого семейства, извлекали свойство `Name` и добавляли в традиционный для LISP список.

При традиционной обработке таблицы блоков код был бы более привычным и более гибким (см. функцию `ai_table` из файла ai_utils.lsp в каталоге %ACAD%\Support), но это только в простом случае с таблицей блоков.

Для преобразования данных из традиционного представления в объектное и обратно необходимо использовать специальные функции, такие как:

- `vlax-ename->vla-object`,
- `vlax-vla-object->ename`,
- `vlax-safearray->list`

и подобные им.

Рекомендуется следующий алгоритм работы с объектной моделью:

1. Изучить объектную модель и найти место интересующего объекта (или семейства) в иерархии объектов.
2. Получить по иерархическому дереву указатель на интересующий объект.
3. Изучить свойства и методы объекта.
4. Получить наименование нужного свойства или метода.
5. Установить нужное свойство или применить нужный метод.

Это укрупненная схема, при реализации нужно учитывать много деталей, например, проверять, доступно ли свойство для изменения.

Мы обычно[1] поступаем следующим образом:

1. В IDE Visual LISP вызываем справку.
2. В разделе "ActiveX and VBA Reference" выбираем требуемый объект (например, Objects > Block Object).
3. На выведенной странице по объекту изучаем его общее описание и списки доступных методов (Methods), свойств (Properties) и событий (Events).
4. Щелкаем по кнопкам в верхней части страницы, чтобы просмотреть, в какие объекты может входить данный объект, какие объекты может содержать данный объект и какие он поддерживает интерфейсы.
5. При необходимости изучаем соответствующие методы и свойства (и примеры, правда, на VBA) в таком же порядке.

Примечание

Дело это, надо сказать, довольно нудное. Нет ощущения рекламируемой легкости. Некоторую помощь оказывает IDE Visual LISP (через окно **Inspect**), но "я такого не хочу даже вставить в книжку", как писал Маяковский. Настолько все сделано убого и неуклюже. Конечно, мы избалованы великолепными инструментами Object Browser и Code Insight из Borland Delphi, но все равно, мы уже живем в XXI веке! Лучше бы AutoCAD 2004 был бы выпущен не на год раньше срока, но с настоящей IDE для Visual LISP. В этом отношении редактор VBA гораздо лучше, в нем имеются хорошие возможности для визуальной разработки программ и исследования объектной модели.

[1] Те читатели, у кого есть книги "AutoCAD 2002" и "AutoCAD 2004: разработка приложений и адаптация" издательства "БХВ-Петербург" в серии "В подлиннике", могут получить аналогичные сведения из глав 43 и 1, соответственно, в которых исправлены некоторые ошибки и упущения справочной системы (особенно это относится к версии 2004).

Помучившись с рекомендованной нами методикой, выработав свою и написав несколько функций, любой разработчик поймет, что работать с объектной моделью не сложно и весьма полезно.

Мы, в свою очередь, в одной из следующих глав сформируем библиотеку высокоуровневых функций, позволяющих работать с объектной моделью *очень легко*. Как в рекламе.

Но использовать объектную модель надо разумно. Можно, например, придумать функцию для установки значений системных переменных:

```
(defun ax_set_sysvar (var_name var_value)
 (vla-SetVariable
  (vla-get-activedocument
   (vlax-get-acad-object)) var_name var_value) var_value)
```

и вызывать ее

```
(ax_set_sysvar "CMDECHO" 1)
```

Однако гораздо проще пользоваться традиционной функцией

```
(setvar "CMDECHO" 1)
```

а метод `SetVariable` оставить для систем программирования, в которых только его и можно применить.

Особенности многодокументного режима

Помимо непривычной объектной модели некоторые сложности вызывают особенности работы в многодокументном режиме. *Многодокументный режим* (MDI — Multiple Document Interface) появился впервые в AutoCAD 2000. Вообще-то это обычный режим работы многих приложений, но не так уж много приложений позволяют программировать для них "изнутри себя". Включение режима MDI производится установкой системной переменной SDI = 0.

Каждый рисунок (в контексте данной темы — документ), открытый в AutoCAD, добавляется в семейство (Collection) `Documents` и имеет собственное пространство имен (NameSpace). *Пространство имен* — это область памяти, а для LISP-программиста это некая среда, в которой изолирован набор используемых им переменных и функций. Переменные и функции, определенные в одном пространстве имен, недоступны в другом пространстве имен. Если мы загрузим функцию или создадим глобальную переменную в одном документе, то, переключившись на другой документ и попытавшись выполнить эту функцию или получить значение переменной, мы получим сообщение об ошибке. Отсюда следует первое правило — *требуемые функции должны загружаться в каждый документ*.

Сумятицу вносит присутствие разных видов "племен" приложений. В AutoCAD используются LISP-приложения, находящиеся в LSP- и FAS-файлах, VLX-приложения, а также приложения, написанные на ObjectARX и VBA. И у каждого свои причуды. Самые демократичные в этом отношении LSP- и FAS-файлы. Функции, определенные в них, после загрузки становятся доступными всем их "соплеменникам".

VLX-приложения, собранные с собственным пространством имен, "не видят" символов, в том числе глобальных, других приложений, в том числе загруженных в этот

же документ. Для передачи символов в пространство документа из таких VLX-приложений следует использовать функции `vl-doc-export` и `vl-doc-set`. Для получения доступа к символам документа в таком VLX-приложении нужно использовать функции `vl-doc-import` и `vl-doc-ref`. В нашей системе мы не будем делать VLX-приложений с собственным пространством имен и не будем более задерживаться на этом вопросе.

Кроме пространств имен каждого документа существует внедокументное пространство (Blackboard). *Внедокументное пространство имен* предназначено для обеспечения доступа к данным из отдельных пространств имен других документов и приложений. В нашей системе мы не будем интенсивно использовать внедокументное пространство, но несколько переменных поместим именно туда. Например, мы будем создавать дополнительную кнопку в статусной строке, и для того, чтобы она не создавалась вновь при инициализации новых документов (статусная строка едина для всех документов), во внедокументное пространство можно поместить переменную, указывающую на то, что кнопка уже создана.

При инициализации системы вызывается функция, создающая кнопку:

```
(ru-acad-sbb-make-button "ruCAD/ACAD" 'ru-acad-sbb-on-click)
```

Внутри функции `ru-acad-sbb-make-button` проверяется значение переменной из внедокументного пространства:

```
(defun ru-acad-sbb-make-button (button_text button_clicked_call_back)
;;; Чтение значения внедокументной переменной
 (if (not (vl-bb-ref '*ru_bb_sbb_made*))
    (progn
      (действия_по_созданию_кнопки)
;;; Установка значения внедокументной переменной
      (vl-bb-set '*ru_bb_sbb_made* T)
   );_ end of progn
 );_ end of if
)
```

Переменная `*ru_bb_sbb_made*` находится во внедокументном пространстве и получить ее значение в любом документе можно только вызовом функции

```
(vl-bb-ref '*ru_bb_sbb_made*)
```

Возможен вариант с использованием функции `vl-propagate`. Эта функция делает доступной глобальную переменную одного документа во всех других открываемых или создаваемых документах. Эта переменная уже не находится во внедокументном пространстве, а ее копии "пропагандированы" во все документы.

```
(defun ru-acad-sbb-make-button2 (button_text button_clicked_call_back)
(if (not *ru_bb_sbb_made*)
    (progn
      (действия_по_созданию_кнопки)
       (setq *ru_bb_sbb_made* T)
       (vl-propagate '*ru_bb_sbb_made*)
   );_ end of progn
 );_ end of if
)
```

При изменении значений переменных в одном документе новые значения необходимо вновь экспортировать или "пропагандировать".

Словари

Словари — это специальные неграфические объекты, сохраняющиеся в DWG-файлах. Система AutoCAD интенсивно использует словари. Все относительно новые возможности AutoCAD базируются на использовании словарей (групп, листов, стилей печати и мультилиний и др.). Еще важнее, что пользовательские приложения могут создавать и использовать свои словари. Подробно работу с пользовательским словарем, в котором сможем сохранять в рисунке любые данные, рассмотрены в *главе 13*.

Реакторы

Реакторы — это специальные объекты, дающие возможность предусмотреть реакцию системы на различные события (изменения базы объектов рисунка, действия с документами, загрузку и выгрузку приложений, изменение конкретных объектов рисунка, выполнение команд и ряд других). Работу с реактором команд мы также рассмотрим в *главе 13*.

Как использовать библиотеки функций и "конечные" программы

Все исходные тексты, которые мы будем разрабатывать, можно разделить на две большие группы — библиотеки функций и программы для "конечных" пользователей. Разумеется, в LISP все является функциями, и даже многие стандартные команды на самом деле функции, которые из других функций можно вызывать в виде (`c:имя_команды`). В нашей системе *LISP-программами* мы будем именовать файлы, содержащие функции, необходимые пользователям для решения их прикладных задач, а *библиотеками LISP-функций* — файлы, содержащие вспомогательные функции, необходимые программистам для разработки программ.

В библиотеках мы будем, по возможности придерживаясь определенной системы, собирать коллекции функций самого разнообразного назначения. Основная работа программиста будет состоять именно в разработке библиотек функций.

Есть два варианта разработки библиотечных функций. Условно, один из них можно назвать "минималистским", другой — "универсалистским". Основное отличие между ними заключается в том, чему отдается приоритет при создании функций — компактности и простоте кода или их функциональной нагрузке и универсальности. Оба подхода имеют свои плюсы и минусы (а кто бы сомневался) — особенно в экстремальных ситуациях.

Для минималистского подхода характерна повышенная надежность каждой, отдельно взятой функции, при большом количестве вызовов функций в готовой программе, постоянной передаче и проверке данных между функциями.

Универсалистский подход позволяет уменьшить как частоту вызовов функций, так и количество самих функций и глобальных переменных. Кроме того, при разработке

небольшого числа крупных функций легче обеспечить единообразие поведения всей программы. Естественно, увеличение функциональности требует дополнительной работы по написанию кода и его отладке. Увеличивается и вероятность ошибок при разработке функций, при этом, нередко, страдает удобство использования таких функций из-за перегруженности аргументами вызова и сложности возвращаемых данных.

Разрабатывать маленькие и простые функции, естественно, проще, чем большие и сложные, но не все так однозначно. Организовать беспроблемное взаимодействие множества маленьких функций — это отдельная и непростая работа. Любой из этих подходов очень легко можно довести до абсурда, что вряд ли положительно скажется на окончательном результате. Оптимальным представляется вариант, когда библиотечная функция решает какую-то одну задачу, но делает это в максимальном многообразии ситуаций. Например, если функция предназначена для запроса у пользователя ввода какого-либо вида данных, то она должна уметь не только "молча" принимать этот вид данных, но и уметь проверять эти данные на ограничения, заданные при ее вызове, с уведомлением пользователя о его ошибках при вводе, а при необходимости и блокировать откровенно "дурацкие", с точки зрения программы, действия. Но совершенно ни к чему, чтобы функция, помимо обработки запроса пользователя, еще бы и что-то рисовала, ее работа — принять данные, проверить и передать "на выход". Что произойдет с данными дальше, не ее дело.

Как и во многих других ситуациях, лучше "золотая" середина. Сбалансированное сочетание простоты и функциональности позволит получить работающий вариант программы быстрее и он будет качественнее, чем при впадении разработчика в любую крайность. Примеры коротких программ приведены в листингах 10.8—10.11.

Листинг 10.8. Пример короткой программы. Вариант 1

```
(defun start ()
 (ru-app-begin)
 (ru-draw-rectangle nil nil nil nil)
 (ru-app-end)
)
(start)
```

Такая программа, с очень коротким исходным текстом, состоящим из вызова трех библиотечных функций, на самом деле делает очень много. В ней есть и диалоговое окно, и много всяких хитростей.

Листинг 10.9. Пример короткой программы. Вариант 2

```
(defun start ()
(ru-app-begin)
 (ru-draw-rectangle "Резервуар прямоугольный" nil nil T)
(ru-app-end)
)
(start)
```

Это уже другая программа, но использующая те же функции. Применяется, очевидно, теми, кто рисует резервуары (генпланистами, топографами, технологами, сантехниками).

Листинг 10.10. Пример короткой программы. Вариант 3

```
(defun start ()
(ru-app-begin)
 (ru-draw-rectangle "Стол полированный" "00000000" "ПР_1" nil)
(ru-app-end)
)
(start)
```

Еще одна программа, использующая те же функции. Явно предназначена для применения архитекторами или технологами, расставляющими оборудование общественных зданий. К тому же возникает подозрение, что имеется еще много программ, также использующих функцию `ru-draw-rectangle`.

Листинг 10.11. Программа-рекордсмен

```
(defun start (block_lib_name block_name type_insert x_mm y_mm is_iso)
(ru-block-insert-from-lib
     block_lib_name block_name type_insert x_mm y_mm is_iso)
(princ)
)
```

Вызов этой программы (листинг 10.11) встречается в наших меню 1 256 раз! Возможно, это программа-рекордсмен, хотя есть еще несколько программ, применяемых с такой же интенсивностью. И даже пользователь класса "нормальный юзер" может, зная параметры функции, легко дополнить собственное меню еще неограниченным количеством пунктов, ведь программировать ему ничего не нужно, надо только уметь рисовать собственные блоки.

Как видим, труд, затраченный на разработку всего одной функции, дает конечный результат, обеспечивающий работу множества пользователей.

> Такие, чисто интерфейсные, ультракороткие "программы", я бы выделил в отдельный тип. Что-нибудь наподобие: "стартеры", "стартовые оболочки" или "меню-скрипты". А программами называл бы как раз те функции, которые эти стартеры запускают. Фактически, никакой полезной нагрузки, кроме переопределения функции `start`, они не несут, а того же результата можно достичь и без создания функции, простой подстановкой. *(Петр Лоскутов)*

Да, действительно, такие короткие куски кода можно постесняться называть программами. Но, назови их хоть как (это всего лишь споры о терминах), они делают возложенную на них работу! Современная программа, написанная на Delphi, обычно содержит всего несколько строк, в которые программист не вписал вручную ни одного символа:

```
begin
  Application.Initialize;
  Application.CreateForm(TFrmStart, FrmStart);
  Application.Run;
end.
```

Вся полезная работа спрятана в формах, модулях, классах. Таковы уж современные подходы к программированию.

Программы-шутки

Создайте файл 0.exe размером 0 байт и "выполните" в различных средах. Судя по имени, это программа, судя по содержанию — ничто. Но это ничтожество заставит "настоящую программу" что-то сделать. Пусть это "что-то" всего лишь раздраженный окрик, возможно даже с диалоговым окном, но результат уже не "ничто", а "нечто". Так программа это, или "что"? А если мы включим в нее пять байт — `MsDos`? А если мы "разработаем" программу 1.com с единственным байтом в виде символа `ы`? Какой потом возможен терминологический спор с "прокурором"!

Как использовать библиотеки сторонних авторов

Для системы AutoCAD разработан ряд библиотек, функции из которых мы могли бы использовать в своей системе. Важнейшей из них для нас является библиотека DOSLib фирмы Robert McNeel & Associates[1]. Эта во всех отношениях замечательная библиотека распространяется с 1992 года, постоянно расширяется и совершенствуется. Версия DOSLib для AutoCAD 2004 появилась одновременно с выходом самого AutoCAD 2004. В библиотеку DOSLib всегда включались именно те функции, которых так не хватает в стандартном AutoCAD, т. е. для работы с операционной системой. Удивляет, почему фирма Autodesk не включает в стандартную поставку аналогичные функции (кое-какие аналоги все-таки появились в Visual LISP) или не лицензирует DOSLib для поставки с AutoCAD. Любой разработчик просто обязан использовать эту библиотеку, тем более что поставляется она бесплатно.

В состав DOSLib входит 161 функция. К сожалению, объем книги не позволяет включить в нее описание всех функций DOSLib, и мы будем вынуждены ограничиваться краткими пояснениями только по используемым нами функциям. В своих прежних разработках мы не использовали DOSLib, т. к. писали свои аналоги. К сожалению, с выходом AutoCAD 2004 положение изменилось. Если когда-то, в добрые старые времена, приложения ADS можно было писать и компилировать в любой среде разработки (для AutoCAD R14 мы это делали даже на Delphi), то теперь это можно делать только с использованием Microsoft Visual C++, да еще только версии, предписанной союзом Autodesk и Microsoft. Для AutoCAD 2002 это *еще* была версия 6, а для AutoCAD 2004 это *уже* версия 7. Даже очень простые ARX-приложения, не использующие никаких особенностей AutoCAD, должны быть, как минимум, просто перекомпилированы в предписанной среде. Так что если вы имеете Microsoft Visual Studio 6, то для создания простого ARX, экспортирующего в LISP несколько функций Windows API, вы уже просто обязаны приобрести Microsoft Visual Studio .NET. Все это вкупе практически губит намерения многих молодых разработчиков, решивших заняться разработкой серьезных приложений для AutoCAD с использованием ObjectARX.

Разумеется, можно почти все, что имеется в DOSLib, реализовать через механизмы ActiveX, и функции, отстутствующие в DOSLib, мы так и будем создавать. Но ARX-функции работают быстрее (нет лишних посредников) и там, где скорость работы имеет значение, мы будем использовать функции DOSLib. Для того чтобы читатели имели примерное представление, как делаются такие библиотеки, мы разработаем собственную ARX-библиотеку с несколькими функциями *(см. главу 14)*.

---

[1] **www.mcneel.com**.

Второй сторонней библиотекой, которую мы, может быть, будем использовать, является библиотека Express Tools.

Про использование программ из библиотеки Express Tools мы уже говорили в *главе 8* и даже предусмотрели механизм управления доступом к командам этой библиотеки. Но Express Tools — это не только сборник полезных программ, но и прекрасная библиотека разработчика. В ней имеется много полезных функций, как в LSP-файлах, так и в файлах acetutil.arx и acetutil.fas. Некоторые функции из acetutil.arx имеют теперь аналоги в Visual LISP, некоторые в DOSLib, некоторые мы напишем сами, но есть и уникальные. К сожалению, мы будем вынуждены ограничиваться краткими пояснениями по функциям, которые будем применять. Для наших программ, использующих функции библиотеки Express Tools, придется также применять блокировку меню. Это не страшно, т. к. это будут не самые необходимые программы.

Кроме указанных библиотек в Интернете можно "разжиться" и другими. Из числа бесплатных стоит упомянуть очень интересную библиотеку STDLIB[1] Рейни Урбана (Reini Urban). Библиотека выложена со всеми исходными текстами, включая CPP-файлы, но вряд ли ей можно воспользоваться в прагматичных целях. Здесь интересны и сам подход к созданию библиотеки сразу для множества платформ, версий AutoCAD, и система оформления исходных текстов (в которой, возможно, кроме автора никто не разберется), и превосходные LISP-функции. Главным отличием функций этого автора и его единомышленников является упор на изящные алгоритмы и особенности LISP как языка. В общем, это чтение для гурманов, хотя поучиться программированию на этих примерах стоит всем.

Как организовать загрузку библиотек и программ

Библиотеки функций должны загружаться в каждый открытый документ. Программы будут загружаться при выборе пользователем пункта из какого-то меню. Некоторые программы, возможно, у конкретного пользователя не будут загружены ни разу, зато другие будут перегружаться каждую минуту. Для некоторых программ могут потребоваться специализированные библиотеки, загрузка которых будет производиться автоматически из самих программ.

В каждый документ загружается файл *acaddoc.lsp*. В этом файле мы и будем записывать все необходимые действия при открытии существующего и создании нового документа. На начальном этапе разработки файл acaddoc.lsp мы будем писать и редактировать вручную. Располагаться он будет в каталоге %ruCadLocalAppDataAcad% *(см. главу 3)*. Изменять и дополнять файл автозагрузки мы будем много раз, поэтому пока приводим его предварительный схематичный вариант (листинг 10.12).

Листинг 10.12. Предварительный вариант файла acaddoc.lsp

```
(princ "\nЗагружаю ruCAD...\n")
;;; Временно создаем глобальную переменную
(setq *ru_root_dir* "c:\\.ru\\cad\\")
```

[1] **http://xarch.tu-graz.ac.at/autocad/stdlib**.

```
;;; Загружаем файлы с функциями, которые нужны сразу
(load (strcat *ru_root_dir* "Source\\Lisp\\Lib\\ru-acad.lsp"))
(load (strcat *ru_root_dir* "Source\\Lisp\\Lib\\ru-doslib.lsp"))
(load (strcat *ru_root_dir* "Source\\Lisp\\Lib\\ru-express.lsp"))
;;; Функция ru-acad-ver определена в ru-acad.lsp
(if (< (ru-acad-ver) 16)
  (progn
    (princ "\nЗагружаю MainLib...\n")
;;; ru_MainLib.arx - наша библиотека, написанная на C++
    (if (not (member "ru_MainLib.arx" (arx)))
      (arxload (findfile "ru_MainLib.arx"))
   )
    (princ "\n......загружена...\n")
;;; Сохраняем признак загрузки нашей библиотеки
    (setq *ru_use_ru_arx* T)
 )
  (progn
    (princ "\n для AutoCAD 2004 работаем с DosLib 2004...\n")
    (setq *ru_use_ru_arx* nil)
  )
)
;;; Загружаем библиотеку Express Tools
(ru-express-load)
;;; Проверяем наличие библиотеки DosLib
(if (or (ru-doslib-load) *ru_use_ru_arx*)
 (progn
;;; Продолжаем загрузку своих библиотек
    (load (strcat *ru_root_dir* "Source\\Lisp\\Lib\\ruax-lib.lsp"))
    (load (strcat *ru_root_dir* "Source\\Lisp\\Lib\\ru-xml.lsp"))
;;; и т. д., все требуемые файлы
;;; После загрузки всех библиотек выполняем инициализацию системы
    (ru-init-start-rucad)
  )
 (alert "Система ruCAD не сможет работать. Не загружена ни MainLib, ни DosLib")
)
```

В таком виде файл acaddoc.lsp будет существовать до тех пор, пока мы не отработаем LISP-библиотеки. На определенном этапе мы будем запускать AutoCAD не с помощью ярлыка, а через программу-стартер *(см. главу 3)*. Программа-стартер, помимо прочих возложенных на нее функций (идентификация пользователя, выбор рабочей версии системы AutoCAD и ее запуск с необходимыми параметрами командной строки), будет формировать файл acaddoc.lsp по специальному шаблону, который позволит, без корректировки программы, вносить необходимые изменения.

Так как стартер знает расположение компонентов нашей системы и параметры AutoCAD, то при обработке шаблона будут произведены замены имен переменных на их значения.

Сгенерированный файл автозагрузки будет удаляться при выходе из системы ruCAD для того, чтобы он не мог случайно загружаться при запуске других систем.

Советы по предотвращению ошибок

Советы, как известно, любят давать все. Советы часто бывают прямо противоположные. В этом разделе мы приведем несколько советов, которые надо выполнять безусловно. Многие из них давно известны разработчикам-практикам, но все-таки слишком часто, судя по нашему опыту, находятся люди, которые, оказывается, не знают, что "перед едой надо мыть руки". Тому, *как надо* делать программы, собственно и посвящена вся эта книга, а вот на некоторые аспекты того, *как не надо* делать, мы обратим внимание в данной главе.

Когда автор разрабатывает и тестирует программу на своем рабочем месте, программа обычно работает. Иногда даже работает правильно. Как только программа оказывается на рабочем месте конечного пользователя, она начинает работать неправильно. Или вообще никак. Как только на это же место садится автор, программа начинает работать или делать вид, что работает. Программисты объясняют это тем, что бестолковые "чайники" неправильно работают с его правильной программой. Да и вообще, мол, здесь "кривая" Windows, "глючный" AutoCAD, а администратор вообще "ламер". Что и говорить, причины "уважительные". Об "неправильных" операционных системах, AutoCAD и администраторах мы уже много рассуждали в *главе 3* и еще будем говорить при обсуждении программы установки ruCAD, пока же разберемся с собственными ошибками, ограничившись только LISP-программами.

Для того чтобы программы содержали минимум ошибок (вообще без ошибок бывают, как известно, только никому не нужные программы), необходимо всегда рассчитывать на самый пессимистичный вариант развития событий. Пользователь будет делать все не так, как надо, AutoCAD будет рисовать не то, что задумано, а вся программа обязательно будет работать по неизвестному вам алгоритму. С учетом специфики САПР можно разделить потенциальные сбойные ситуации на ошибки ввода данных, ошибки рисования, обработки данных и ошибки алгоритма. Помимо этих, могут существовать и другие ошибки, но мы попытаемся разобраться с основными.

Ошибки этапа ввода данных

Программист должен рассчитывать на то, что пользователь будет делать неправильно *все*, что только можно. Запрещать ему делать *все вообще*, к сожалению, нельзя, значит надо организовать процедуру запроса данных так, чтобы "правильные" действия были совершенно очевидны и, по возможности, как можно меньше "нагружали" добросовестного пользователя, а "особо одаренных" — *заставить* делать только правильно. Но и пользователь вправе рассчитывать, что программа написана правильно, надежно и "для него".

Контроль ввода

Часто в программах бывает написано что-то наподобие:

```
(setq pt1 (getpoint "\nНачало линии: "))
(setq pt2 (getpoint pt1 "\nКонец линии: "))
```

Если пользователь *не укажет* первую точку, произойдет ошибка и вина здесь не пользователя, а программиста, не предусмотревшего контроль ввода данных.

```
Начало линии:
; error: bad argument type: point: nil
```

Правильно нужно делать так:

```
(while (setq pt1 (getpoint "\nНачало линии <Выход>: "))
 (initget 1)
 (setq pt2 (getpoint pt1 "\nКонец линии: "))
 (command "_.LINE" pt1 pt2 "")
;;; Другие действия
)
```

В этом варианте пользователь *может* указать первую точку, а может и завершить работу, нажав клавишу <Enter>, и он знает об этом, видя результат возможного пустого ввода, который показан в принятой в AutoCAD форме. Вторую точку пользователь уже *обязан* указать. Для этого перед `getpoint` выполнена функция `initget` с битом 1, запрещающим пустой ввод.

Функцию `initget` программист должен прекрасно знать и применять часто, но не всегда. Иногда программист знает про функцию `initget`, но совершает другую ошибку — заставляет пользователя вводить много данных, не давая штатной возможности для выхода. Встречаются и совсем экстремальные случаи, когда на таком приеме (прерывания программ только нажатием клавиши <Esc>) построены целые "системы":

```
(initget 1)
(setq pt1 (getpoint  "\nНачало линии: "))
(while pt1
  (initget 1)
  (setq pt2 (getpoint pt1 "\nСледующая точка: "))
  (command "_.LINE" pt1 pt2 "")
  (setq pt1 pt2)
;;; Другие действия
)
```

Правильно нужно делать так:

```
(while (setq pt1 (getpoint "\nНачало линии <Выход>: "))
(while(setq pt2 (getpoint pt1 "\nСледующая точка <Достаточно>: "))
  (command "_.LINE" pt1 pt2 "")
  (setq pt1 pt2)
;;; Другие действия
)
)
```

В этом варианте пользователь имеет возможность выйти как не указывая первую точку, так и во время запроса любой точки. Если точки не указаны, перехода к рисованию просто не происходит. Этот пример можно написать иначе, убрав цикличный запрос первой точки:

```
(if (setq pt1 (getpoint "\nНачало линии <Не надо>: "))
(while(setq pt2 (getpoint pt1 "\nСледующая точка <Достаточно>: "))
  (command "_.LINE" pt1 pt2 "")
  (setq pt1 pt2)
;;; Другие действия
)
)
```

Какой вариант лучше, зависит от особенностей применения программы. Но явное указание на выход должно быть всегда, особенно если выходом не является пустой ввод.

Ввод точек (как и других данных) надо предусматривать с возможностью передачи значения по умолчанию, кроме случаев, когда значений "по умолчанию" в принципе не может быть, исходя из логики работы программы. Вообще, идеологию функций ввода данных всегда лучше разрабатывать в первую очередь, т. к. они будут применяться очень много раз.

Ввод чисел требует особого внимания. Числа могут быть действительными и целыми, для числа может задаваться диапазон допустимых значений. Тип введенных данных должен проверяться до того, как данные будут использоваться. Иногда это требует значительных усилий со стороны разработчика. Например, одно из основных свойств рисунка в системе ruCAD — масштаб чертежа отображается и может быть изменено в диалоговом окне команды DWGPROPS (СВОЙСТВАРИС). На правильность ввода в этом окне мы не можем оказать никакого влияния, кроме как припугнуть пользователя. Мы вынуждены сделать реактор команд, который отслеживает выполнение команды DWGPROPS (СВОЙСТВАРИС), до ее начала запоминает прежние значения, после окончания проверяет корректность введенных значений, а если они неправильные, то извещает пользователя и восстанавливает прежние корректные данные.

Ввод углов осложняется тем, что углы могут быть в градусах или радианах. Пользователь обычно вводит угол в градусах, а в программе должны производиться необходимые преобразования. Если программист при тестировании все время вводит угол 0, он может не заметить неверного преобразования.

Любые функции ввода следует проверять на ввод заведомо неверных данных!

Опасайтесь двусмысленных предложений

Даже квалифицированного пользователя могут превратить в "чайника" неясные пункты меню и приглашения для ввода данных. Допустим, пользователь выбирает в меню пункт "Трубопровод" (хорошо, если не "Труба"), и получает первый запрос:

```
Диаметр:
```

Девушка из "поколения пепси", не обремененная знаниями, может просто написать "100" и ждать, что произойдет.

Сам программист, заранее знающий, что это всего-навсего подпись диаметра, введет "qwert" и будет демонстрировать потрясающие возможности программы, автоматически добавляющей к *любому тексту* знак диаметра.

"Тетка", все знающая про трубы, задумается над тем:

- что вообще делает эта программа — рассчитывает сопротивление труб, выполняет расчет на прочность, рисует трубу "в три линии", а может быть схематично, и какой трубопровод (вода?, газ?, пар?, сжатый воздух?) имеется в виду;
- какой диаметр надо вводить — условного прохода, наружный, внутренний, в каких единицах (в одной программе приходилось вводить непременно в метрах), надо ли указывать толщину стенки, а если надо, то как (то ли еще будет вопрос, то ли написать через русскую букву "х", то ли через "*", то ли через английскую "x").

Как вы думаете, кто в такой ситуации будет считаться "неправильным" и кто такой (мягко говоря) на самом деле? А что стоило автору написать в меню "Подпись диаметра трубопровода", предусмотреть строку подсказки (helpstring) "Подписывает диаметр трубопровода с добавлением знака диаметра перед любым текстом" и вывести запрос: "Введите текст диаметра [Справочник] <Выход>: "?

Не стоит в этом случае ссылаться на то, что "тетки" не читают документацию. Если начать "разбор полетов", то скорее всего выяснится, что и нормальной документации у вас нет.

Ввод строк вроде бы самый простой (текст может быть любым), но даже при использовании функции getstring программисты совершают единственную возможную ошибку — забывают, что для возможности ввода пробелов в строке нужно указывать флаг пробела:

```
(getstring T "Введи строку с пробелами: ").
```

Эта *детская* ошибка часто происходит из-за того, что программисты ленятся вводить настоящие данные, а набирают на клавиатуре что-то вроде "аааааааа".

Выбор объектов и создание наборов примитивов

Это также потенциальный источник ошибок. Не следует рассчитывать на то, что пользователь в ответ на приглашение "Выбери отрезок:" укажет, как предполагал автор программы, именно примитив LINE. И не только из-за "вредности", но и потому, что может не знать, на что он указывает — действительно на отрезок, или на что-то *похожее на отрезок*. Да и вообще он имеет право "промазать". Предусматривать все ситуации должен программист. Если требуются примитивы только определенных типов, следует применять фильтры. В общем случае набор примитивов создается с помощью функции ssget. Набор можно создать указанием примитивов с использованием таких же опций, какие используются в диалоге с командами AutoCAD и с заданием условий, которым должны удовлетворять выбранные примитивы. Фильтр примитивов задается ассоциированным списком, использующим DXF-коды.

Мы не будем описывать всех возможностей функции ssget, просто приведем несколько примеров.

Выбор всех отрезков:

```
(setq selection (ssget "_x" '((0 . "LINE"))))
```

Примечание

Обратите внимание — метод выбора мы указали с префиксом "_"! Об этом мы уже не раз предупреждали.

Выбор всех отрезков на слое WALL:

```
(setq selection (ssget "_x" '((0 . "LINE")(8 . "WALL"))))
```

Имя слоя является константой. А если требуется переменная? *Ошибочным* будет код:

```
(setq layer_name "WALL")
(setq selection (ssget "_x" '((0 . "LINE")(8 . layer_name))))
```

Примечание

Передать имя слоя в виде переменной в фильтр нельзя. Необходимо создавать точечную пару с помощью функции cons, а список формировать функцией list.

Выбор всех отрезков на слое, заданном переменной. Правильно:

```
(setq layer_name "WALL")
(setq selection (ssget "_x" (list '(0 . "LINE")(cons 8 layer_name))))
```

Выбор всех примитивов на слое, заданном переменной:

```
(setq selection (ssget "_x" (list (cons 8 layer_name))))
```

Выбор всех отрезков и компактных полилиний:

```
(setq selection (ssget "_x" '((-4 . "<OR")(0 . "LINE")(0 . "LWPOLYLINE")
(-4 . "OR>"))))
```

В этом примере в группе −4 был задан групповой логический оператор OR. А можно написать проще:

```
(setq selection (ssget "_x" '(0 . "L*INE")))
```

Выбор всех отрезков на слое WALL и компактных полилиний на слое DOOR:

```
(setq selection (ssget "_x" '((-4 . "<OR")
                              (-4 "<AND")
                                (0 . "LINE")(8 . "WALL")
                              (-4 "AND>")
                              (-4 "<AND")
                                (0 . "LWPOLYLINE")(8 . "DOOR")
                              (-4 "AND>")
                             (-4 . "OR>"))
               )
)
```

В реальных программах необходимости в подобных фильтрах обычно нет, но возможности знать надо.

Выбор полилиний:

```
(setq selection (ssget '((-4 . "<OR")(0 . "POLYLINE")(0 . "LWPOLYLINE")
(-4 . "OR>"))))
```

В данном примере могут оказаться выбранными также полигональные и многогранные сети, которые тоже являются примитивами POLYLINE.

Выбор всех примитивов, которые можно редактировать как полилинии:

```
(setq plines (ssget "_:L";;Исключаем примитивы на заблокированных слоях
'(
        (-4 . "<OR")
         (0 . "ARC")
         (0 . "LINE")
         (-4 . "<AND")
          (0 . "POLYLINE")
          (-4 . "<NOT")
           (-4 . "&") (70 . 88)
```

```
          (-4 . "NOT>");8 16 64 not a 3dpoly or 3d/pface mesh
         (-4 . "AND>")
         (0 . "LWPOLYLINE")
        (-4 . "OR>")
       )
     )
)
```

Примечание

Для предотвращения ошибок, связанных с выбором (и последующей попыткой обработки) примитивов на заблокированных слоях мы использовали метод `(ssget "_:L")`. Попробуйте найти его в документации.

Выбор всех примитивов на текущем слое, у которых имеются расширенные данные приложения RUCAD:

```
(setq selection (ssget "_x" (list (cons 8 (getvar "CLAYER"))
                                  (list -3 (list "RUCAD")))))
```

Символьные имена, задаваемые в списках фильтров — тип примитива (код 0), имя блока (код 2), имя слоя (код 8), тип линии (код 6) и т. п. могут содержать шаблоны глобальных символов, такие же, как и в функции `wcmatch`.

Выбор на слоях: DOOR, WALL_1, WALL_2, WALL_3:

```
(setq selection (ssget "_x" '((0 . "DOOR")(1 . "WALL_#"))))
```

Выбор отрезков на слоях: WALL, WINDOW_1FRAME, WINDOW_2FRAME, WINDOW_9F, DIM_AR, DIM_WATER:

```
(setq selection (ssget "_x" '((0 . "LINE")
                (8 . "WALL,WINDOW_[1-9]F*,DIM_@*"))))
```

В этом примере заданы допускаемые имена слоев:

- "WALL";
- начинающиеся с "WINDOW_", продолжающиеся символом в диапазоне от 1 до 9 и заканчивающиеся любой последовательностью символов, начинающейся с F;
- начинающиеся с "DIM_", продолжающиеся любым буквенным символом и заканчивающиеся любой последовательностью символов.

Не будут выбраны отрезки на слоях: DOOR, WINDOW, WINDOW_1GLASS, DIM_1_AR, DIMENSION.

Необходимо предотвращать выбор примитивов не в текущем пространстве. Это может произойти при использовании методов "A" (All) или "P" (Previous).

Предотвратить такой выбор может функция `ru-ss-current-space-filter` (листинг 10.13).

Листинг 10.13. Функция `ru-ss-current-space-filter`

```
(defun ru-ss-current-space-filter ()
(if (= 1 (getvar "cvport"))
  (list '(67 . 1) (cons 410 (getvar "ctab")));;Лист
  (list '(67 . 0));;Модель
))
```

В этой функции используются DXF-коды 67 (1 — пространство листа, 2 — модели) и 410 — текущий Layout.

Примеры использования:

```
(setq filter (ru-ss-current-space-filter))
(ssget filter)
(ssget (append '((0 . "LINE")) filter))
(ssget "_:L" filter)
```

Иногда нужно составить набор из примитивов, созданных после известного нам примитива. Это можно сделать с помощью функции `ru-ss-select-after-ent` (листинг 10.14).

Листинг 10.14. Функция `ru-ss-select-after-ent`

```
(defun ru-ss-select-after-ent (ent_name / selection)
(if (not ent_name)
 (setq selection (ssget "_X"))
 (progn
  (setq selection (ssadd))
  (while (setq ent_name (entnext ent_name))
    (ssadd ent_name selection)
 )
  (if (zerop (sslength selection))
    nil
    selection
 )
)
)
)
```

Для облегчения и унификации выбора объектов, а также для локализации возможных ошибок, мы разработаем специальную группу функций для формирования наборов и работы с ними. Это будут и простые функции, наподобие выбора по имени блока или слоя по имени, и более сложные. Полезные примеры можно найти в исходных текстах библиотеки Express Tools.

Выбор одного примитива

Выбор одного примитива указанием делается просто — функцией `entsel`. Сложности возникают в случае, если необходим определенный тип примитива. Фильтр в функции `entsel` применить нельзя. Обычно тип примитива проверяют в основной функции.

```
(while (setq ent (entsel
      "\nВыбери блок для просмотра описания  <Хватит>: "))
      (setq edata (entget (car ent)))
      (if (= "INSERT" (cdr (assoc 0 edata)))
        (progn
           (setq block_name (cdr (assoc 2 edata)))
           (setq block_description (cdr (assoc 4 (entget
```

```
            (tblobjname "BLOCK" block_name)))))
            (if (not block_description) (setq block_description ""))
            (princ (strcat "\nБлок: " block_name ", пояснение: "
                                    block_description "\n"))
      )
  )
```

Эта обычная практика, но мы постараемся усовершенствовать и эту операцию (листинг 10.15).

Листинг 10.15. Функция `ru-ss-entsel-by-type`

```
(defun ru-ss-entsel-by-type
       (message msg_err_types list_types  / ent ent_type bad_type)
;|
Параметры:
message — приглашение для выбора
msg_err_types — сообщение об ошибочном типе выбранного объекта
list_types — список имен допустимых типов примитивов
Пример:
(if (setq ent_name (ru-ss-entsel-by-type "Выбери отрезок или полилинию"
 "Это не ОТРЕЗОК и не ПОЛИЛИНИЯ" (list "LINE" "LWPOLYLINE")))
(setq new_ss (ru-ss-select-after-ent ent_name))
)
|;
(setq bad_type T)
(while (and bad_type (setq ent (ru-ss-entsel message)))
  (setq ent_type (cdr (assoc 0 (entget ent))))
  (cond ((not (member ent_type list_types))
         (princ (strcat "\nОШИБКА: Указан объект типа '" ent_type
                        "'. " msg_err_types))
        )
         ((ru-layer-is-locked (cdr (assoc 8 (entget ent))))
           (princ "\nОШИБКА: Объект на заблокированном слое!")
        )
         (T (setq bad_type nil))
 )
)
ent
);_ end of defun
```

В этой функции мы применили еще несколько приемов, на которые нужно обратить внимание. Вот как она работает:

- пользователю предлагается выбрать примитив, при этом можно "сказать", какие типы примитивов допустимы;
- при выборе примитива пользователь имеет право промахнуться, при этом работа не будет прервана;
- пользователь может явно отказаться от выбора, нажав клавишу <Enter>;

- если пользователь укажет на примитив, находящийся на заблокированном слое, ему достаточно тактично об этом будет сказано;
- если будет указан примитив недопустимого типа, он получит информацию о типе указанного объекта и, дополнительно, то, что вы пожелаете ему по этому поводу добавить.

Внутри функции производится вызов других функций (листинг 10.16).

Листинг 10.16. Функция `ru-ss-entsel`

```
(defun ru-ss-entsel (message / ent)
;;; Пример: (ru-ss-entsel "Выбери объект, но не промахнись!")
(setvar "ERRNO" 0)
(while
  (and
    (not (setq ent (car (entsel (strcat "\n" message "<Выход>:")))))
    (equal 7 (getvar "ERRNO"));;Ошибка указания при выборе
  )
  (setvar "ERRNO" 0);; сбрасываем значение системной переменной ERRNO
)
(cond
  ((equal (getvar "ERRNO") 52);; пустой ответ
     nil
 )
  (T ent)
))
```

В этой функции обрабатывается интересный вариант использования функции `entsel`, связанный с тем, что пользователь может или нажать клавишу <Enter>, или просто промахнуться при указании примитива. Функция `entsel` и в том, и в другом случае вернет `nil`, но по разным причинам, которые нам хочется знать. До истины причины мы добираемся путем анализа системной переменной ERRNO, которая в случае ошибочного указания принимает значение 7, а при пустом вводе принимает значение 52. Запрос повторяется в цикле, пока не будет нажата клавиша <Enter> или не будет указан примитив (если будет нажата клавиша <Esc>, то произойдет ошибка и "вылет" не только из этой функции, но и из всей программы). Обратите внимание на то, что мы применили функцию `cond`, а не `if` для проверки результата на выходе из цикла. Функция `cond` позволяет проверить несколько условий последовательно. При необходимости мы сможем расширить эту функцию анализом других значений ERRNO. На выходе функции `ru-ss-entsel` мы имеем или имя примитива, или `nil`.

Возвращаемся к функции `ru-ss-entsel-by-type`. После получения имени примитива мы проверяем, входит ли тип примитива в список допустимых, не находится ли он на заблокированном слое. Если все правильно, функция вернет имя примитива, но уже "проверенное на вшивость".

Для проверки блокировки слоя объекта используется функция `ru-layer-is-locked` (листинг 10.17). Она проста, но достойна рассмотрения.

Листинг 10.17. Функция `ru-layer-is-locked`

```
(defun ru-layer-is-locked (layer_name)
(ru-match-is-bit-in-flag 4 (ru-layer-get-ent-data layer_name 70))
)
```

Работа этой функции заключается в проверке условия "входит ли число 4 (флаг блокированного слоя) в сумму битов, ассоциированную с групповым кодом 70 слоя `layer_name`". Сумма битов возвращается функцией `ru-layer-get-ent-data` (листинг 10.18), а проверка вхождения флага — функцией `ru-match-is-bit-in-flag` (листинг 10.19).

Листинг 10.18. Функция `ru-layer-get-ent-data`

```
(defun ru-layer-get-ent-data (name code / lst)
;; возвращает данные с кодом CODE для слоя NAME
;; если слой не существует - nil
(if (setq Lst (tblsearch "LAYER" name)) (cdr (assoc code lst)) nil)
)
```

Листинг 10.19. Функция `ru-match-is-bit-in-flag`

```
(defun ru-match-is-bit-in-flag (val flag)
;;(ru-match-is-bit-in-flag  8 127) > T
;;(ru-match-is-bit-in-flag  128 127) > nil
(= (logand val flag) val)
)
```

А зачем такие сложности? Вместо

```
(ru-layer-is-locked (cdr (assoc 8 (entget ent))))
```

можно было просто написать

```
(= (logand 4
   (cdr (assoc 70 (tblsearch "LAYER" (cdr (assoc 8 (entget ent)))))))
 4)
```

Конечно, можно. Если бы мы писали единственную или первую программу, мы так бы и сделали. Вообще не стали бы придумывать функцию `ru-ss-entsel-by-type`, а написали бы все прямо в теле программы. Про возможность блокировки слоев мы могли бы и не вспомнить. Потом, когда бы стали писать десятую программу, разыскали бы в первой подходящую конструкцию и скопировали. А через год "вдруг" обнаружили бы, что пользователи имеют привычку блокировать слои и забывать их разблокировать при работе с нашими замечательными программами. Нам пришлось бы исправлять недочеты уже во многих местах.

Представьте себе, сколько раз в программах потребуется проверка блокировки слоя. Каждый раз надо будет писать подобное выражение (с немалой вероятностью ошибки).

А вы уверены, что знаете, что означают цифры 4, 8 и 70? А что делает функция `logand`? А если нам еще потребуется проверять слой на "заморозку"? На сочетание

заморозки и блокировки? И вообще, сможете ли вы "с ходу" разобраться, как работает такое выражение? Все подобные действия мы инкапсулировали в простые короткие функции, изменив любую из них, мы избегаем внесения изменений в десятки программ. Каждая из этих маленьких функций будет использоваться и в других программах. Маленькая функция `ru-match-is-bit-in-flag` отвечает на вопрос, наподобие: "Входит ли бит 4 в значение, переданное функции `initget`?". Вы уверены, что знали раньше, как на такой вопрос отвечать?

> А это уже впадение в "минимализм" — уж `logand` и работу с DXF-кодами LISP-разработчик обязан знать, а конкретные значения проще посмотреть в справке, чем вспоминать (или искать) такие "микрофункции". *(Петр Лоскутов)*

Можно еще больше раздробить функции. Получение имени слоя по коду 8 в наших программах встречается 148 раз. Мы могли бы все конструкции `(cdr (assoc 8 (entget ent)))` заменить на вызов функции

```
(defun ru-ent-get-layer (ent)
(cdr (assoc 8 (entget ent)))
)
```

Сейчас мы получили свойство слоя через DXF-коды. Если мы решим окончательно перейти на технологию ActiveX, мы просто *в одном* месте изменим текст функции.

```
(defun ru-ent-get-layer (ent)
(vla-get-layer (vlax-ename->vla-object ent))
)
```

Теперь все наши программы, использующие эту функцию, перейдут на объектную модель.

Мы специально очень подробно разобрали этот пример, чтобы читатели усвоили основные принципы разработки библиотек системы ruCAD. Мы постоянно будем создавать множество маленьких функций, позволяющих легко разобраться с тем, как они работают. Из множества маленьких функций мы будем создавать маленькие (по размеру) программы.

> Такой подход действительно можно критиковать. При первом знакомстве с нашими библиотеками действительно трудно понять логику программ, написанных как бы на незнакомом диалекте LISP. С таким сталкиваются не только в LISP. В дальнейшем мы будем применять множество процедур и функций для Delphi, используя сторонние библиотеки. Да, разобраться, где и что лежит в чужой библиотеке и как это применять, бывает сложнее, чем вписать "по месту" аналогичный участок кода. Тем не менее, мы делаем именно так, потому что пишем очень много программ и стремимся локализировать возможные точки возникновения ошибок. *(Сергей Зуев)*

Мы специально приводим в разделе, посвященном ошибкам, наши внутренние "разборки", чтобы читатели еще раз поняли, что однозначно правильных решений может и не быть. Подход, ошибочный в одной ситуации, может оказаться правильным в другой. Ошибкой будет зацикленность на каком-то одном решении.

Ошибки этапа обработки данных и рисования

После того как мы, по возможности, предотвратили ошибки ввода данных, надо позаботиться об этапе обработки введенных данных. На этом этапе обычно производятся различные вычисления, а затем рисование или модификация объектов. Сна-

чала разберемся с вычислениями. Здесь самой популярной является функция `setq`, соответствующая *операторам* присваивания в других языках[1].

При всей кажущейся простоте вопроса нам необходимо разобраться с переменными и их значениями. Неправильное присваивание может являться источником трудно обнаруживаемых ошибок.

Переменные и значения переменных

Сначала мы не планировали затрагивать в книге такие элементарные понятия, но затем, столкнувшись с тем, что даже опытные программисты владеют ими слабо, решили внести краткие пояснения.

В LISP (как языке) вообще и в AutoLISP в частности существуют понятия "переменная" и "значение переменной". *Переменная*[2] — указатель на адрес в памяти, которому присвоено имя. *Значение переменной — данные*, записанные в динамической памяти, начиная с адреса, записанного в *переменной*. Любые манипуляции с переменной осуществляются обращением к ее символьному имени.

Значение переменной обычно изменяется с помощью функции `setq`, которой передается список пар "символ значение" (в частном случае только одна пара). Присваивание значений с помощью функции `setq` позволяет не знать об организации памяти. До поры до времени.

Сама переменная изменяется с помощью функции `set`. При изменении *переменной* она будет указывать на другой участок памяти, а при изменении *значения переменной* в прежнем участке памяти будут размещены другие данные.

Если переменной присвоить значение `nil`, то сама она сохранится, но будет указывать в никуда, т. е. освободит память, прежде занятую ее значением.

А как же создаются переменные? Допустим, мы в командной строке введем

```
(setq x 1)
```

Тем самым мы выделили участок памяти, присвоили ему имя `X` и поместили туда значение `1`. Проверим:

```
!x
1
```

А теперь вы вспомнили, что читали какое-то учебное пособие, где сказано: "`(set a b)` заставляет переменную `a` ссылаться на ту же область памяти, что и переменная `b`", и пробуете это на практике:

```
Command: (set y x)
; error: bad argument type: symbolp nil
```

Примечание

В AutoCAD R14 это выглядело бы так:

```
Команда: (set y x)
ошибка: неверный тип аргумента
(SET Y X)
```

Был бы еще выведен текст ошибочного выражения.

---

[1] Кстати, вы не забыли, что в LISP операторов вообще нет, используются только функции?

[2] В соответствии с документацией термины "переменная" и "символ" в AutoLISP взаимозаменяемы.

Что за ерунда? Неужели "доценты с кандидатами" ничего не проверяют? Пробуем пример из документации по AutoCAD (специально пользуемся русской версией R14 для более ясных сообщений):

Первая проба функции `set`:

```
Команда: (set b 640)
ошибка: неверный тип аргумента
(SET B 640)
*Отмена*
```

Опять вместо указанного "вернет 640" возвращается ошибка!

Давайте все-таки выполним примеры по порядку:

Вторая проба функции `set`:

```
Команда: (set 'a 5.0)
5.0
Команда: (set (quote b) 'a)
Команда: (set b 640)
640
```

Все-таки работает! Но в чем же дело? А "дело, видите ли, в том, что" первый раз мы пытались присвоить *символу b значение выражения 640.* В тот момент у нас не существовал символ `b`, т. е. мы пытались присвоить значение несуществующей переменной. При второй пробе мы, действуя по порядку, создали переменную `a`, присвоили ей значение 5.0, затем создали переменную с именем `b` и присвоили ей значение выражения `a`. Но еще, поскольку в `(set b 640)` имя переменной было указано без апострофа, мы косвенно изменили значение переменной `a`, в чем легко убедиться:

```
Команда: !a
640
```

Теперь у нас есть переменная `b` и мы можем ей присваивать другие значения

```
Команда: (set b 710)
710
Команда: (set b c)
nil
```

Понятно, переменной `c` не существует, `b` стала "указывать в никуда", а что у нас происходит с `a`? Продолжаем эксперименты:

```
Команда: !a
nil
Команда: (set b "Это B")
"Это B"
Команда: !a
"Это B"
```

Видите, до чего доэкспериментировались? Изменяем одну переменную, а меняется еще и другая. Это мы сейчас все помним и видим, а *в реальной программе мы могли бы долго гадать, почему происходят необъяснимые вещи*. Давайте, пожалеем переменную `a` и "отцепим" от нее переменную `b`:

```
Команда: (setq b "Расходимся с A!")
"Расходимся с A!"
Команда: !a
"Это B"
```

Так зачем же нужна эта "вредная" функция `set`? Что это за таинственные случаи, про которых в книгах пишут "иногда необходимо"? Вот пример нашей будущей функции (листинг 10.20).

Листинг 10.20. Функция `ru-var-toggle-var`

```
(defun ru-var-toggle-var (svar)
(set (read svar)
   (not (eval (read svar)))
);_ end of set
;;Удален неинтересный сейчас участок функции
(princ)
);_ end of defun
```

Это универсальная функция-переключатель значения переменных, имеющих допустимые значения `T` или `nil`. Нам нужно передать *имя переменной*, а функция должна изменить *значение этой переменной* на противоположное. Заодно пример иллюстрирует работу функций `eval` и `read`. Попробуйте такое сделать на другом языке — передать имя переменной в виде строки и присвоить этой переменной значение (имя-то мы еще могли бы "сконструировать").

Протокол работы переключателя

```
(setq *debug* T)
(ru-var-toggle-var *debug*)
!*debug*
nil
(ru-var-toggle-var *debug*)
!*debug*
T
```

Разберем подробно, что делает функция `ru-var-toggle-var`. Мы передаем ей имя переменной в виде строки. "Ухудшим" эту функцию, введя локальные переменные, в которых можно отслеживать значения на разных этапах выполнения.

"Ухудшенная", но понятная функция `ru-var-toggle-var`:

```
(defun ru-var-toggle-var-1 (var_name / var_read var_eval var_not)
(princ "\nВ функцию передан параметр VAR_NAME со значением ")
(print var_name)(princ " и типом ")(print (type var_name))
(setq var_read (read var_name))
(princ
(strcat[1] "\nК переменной VAR_NAME применена функция READ"
"\n(setq var_read (read var_name))"
```

[1] `strcat` мы применяем только для того, чтобы длинные сообщения поместились на странице книги.

```
"\nи получена переменная VAR_READ со значением "))(print var_read)
(princ " и типом ")(print (type var_read))
(setq var_eval (eval var_read))
(princ
(strcat "\nК переменной VAR_READ применена функция EVAL "
"\n(setq var_eval (eval var_read))"
"\nи получена переменная VAR_EVAL со значением "))(print var_eval)
(princ " и типом ")(print (type var_eval))
(setq var_not (not var_eval))
(princ
(strcat "\nК переменной VAR_EVAL применена функция NOT "
"\n(setq var_not (not var_eval))"
"\nи получена переменная VAR_NOT со значением "))(print var_not)
(princ " и типом ")(print (type var_not))
(set var_read var_not)
(princ
(strcat "\nСимволу VAR_READ присвоено значение переменной VAR_NOT"
"\n(set var_read var_not)"
"\nЗначение переменной VAR_NAME на выходе: "))(print var_name)
(princ " с типом ")(print (type var_name))
(princ)
)
```

Напишем, выполним и проанализируем контрольный пример (листинг 10.21).

Листинг 10.21. Функция `test-toggle-var`

```
(defun test-toggle-var ()
(princ "\n\nЗначение переменной *DEBUG* до 1-го переключения:")
(print *debug*)
(ru-var-toggle-var-1 "*debug*")
(princ "\n\nЗначение переменной *DEBUG* после 1-го переключения:")
(print *debug*)
(ru-var-toggle-var-1 "*debug*")
(princ "\n\nЗначение переменной *DEBUG* после 2-го переключения:")
(print *debug*)
(princ)
)
```

Протокол работы тестовой функции (исключены лишние переводы строк)

```
Значение переменной *DEBUG* до 1-го переключения: T
В функцию передан параметр VAR_NAME со значением
"*debug*" и типом STR
К переменной VAR_NAME применена функция READ
(setq var_read (read var_name))
и получена переменная VAR_READ со значением *DEBUG*  и типом SYM
К переменной VAR_READ применена функция EVAL
(setq var_eval (eval var_read))
и получена переменная VAR_EVAL со значением T и типом SYM
```

```
К переменной VAR_EVAL применена функция NOT
(setq var_not (not var_eval))
и получена переменная VAR_NOT со значением nil и типом nil
Символу VAR_READ присвоено значение переменной VAR_NOT
(set var_read  var_not)
Значение переменной VAR_NAME на выходе: "*debug*" с типом STR

Значение переменной *DEBUG* после 1-го переключения: nil
В функцию передан параметр VAR_NAME со значением
"*debug*"  и типом STR
К переменной VAR_NAME применена функция READ
(setq var_read (read var_name))
и получена переменная VAR_READ со значением *DEBUG* и типом SYM
К переменной VAR_READ применена функция EVAL
(setq var_eval (eval var_read))
и получена переменная VAR_EVAL со значением nil и типом nil
К переменной VAR_EVAL применена функция NOT
(setq var_not (not var_eval))
и получена переменная VAR_NOT со значением T и типом SYM
Символу VAR_READ присвоено значение переменной VAR_NOT
(set var_read  var_not)
Значение переменной VAR_NAME на выходе: "*debug*" с типом STR

Значение переменной *DEBUG* после 2-го переключения:T
```

Проанализируйте этот, до предела "разжеванный" пример, сделайте свой, и вы навсегда запомните особенности присваивания значений переменным и будете легко применять великолепные функции `read` и `eval`.

Функция `eval` вычисляет список, являющийся ее аргументом, и возвращает вычисленное значение. Ее использование позволяет в ходе выполнения программы сформировать список-программу (содержащий вызовы функций) и исполнить его.

Зачем введена функция *quote*?

Мы уже писали, что в языке LISP нет различия между вычисляемыми выражениями и обрабатываемыми данными. И выражения, и данные являются списками и, по умолчанию, любой список считается выражением. Но если в списке не выражение, а данные, то надо каким-то образом пресечь попытки интерпретатора "выполнить" список данных. Допустим, в списке находятся координаты точки, и мы пытаемся присвоить переменной `point_1` значение координат, записав их в виде списка:

```
(setq point_1 (0.0 0.0))
Command: (setq point_1 (0.0 0.0))
; error: bad function: 0.0
```

Интерпретатор LISP воспринял список как вычисляемое выражение, а его первый элемент как имя функции, которой он не знает. Для того чтобы предотвратить попытки вычислений и введена функция `quote`, запрещающая вычисление списка и возвращающая сам этот список. Так как данные в виде списка встречаются часто, введена сокращенная запись функции `quote` в виде апострофа:

```
Command: (setq point_1 (quote (0.0 0.0)))
(0.0 0.0)
Command: (setq point_1 '(0.0 0.0))
(0.0 0.0)
```

Контроль типов данных

Язык LISP относится к группе бестиповых языков. Конечно, значение каждой переменной имеет какой-то тип данных, но этот тип может меняться во время выполнения. Выяснить тип данных можно с помощью функции `type`. LISP использует следующие типы данных:

- символы (SYM) — переменные;
- списки (LIST);
- строковые константы (STR);
- целые числа (INT) — числа, не содержащие дробной части;
- вещественные числа (REAL) — числа с плавающей точкой двойной точности;
- "имена" примитивов (ENAME) — числовые метки, присваиваемые примитивам рисунка и возвращаемые, например, функцией `entlast`;
- наборы (PICKSET) — наборы из одного и более примитивов;
- встроенные функции (SUBR) — внутренние функции, которые могут быть переопределены с помощью `defun`;
- пользовательские функции (USUBR) — LISP-функции, определенные через '`defun`', т. е. те, которые могут быть созданы пользователем;
- внешние функции (EXRXSUBR) — функции, определенные в ARX-приложениях;
- дескрипторы файлов (FILE) — указатели на открытые файлы;
- VLA-объекты (VLA-OBJECT) — объекты, используемые в объектной модели;
- варианты (VARIANT);
- безопасные массивы (SAFEARRAY);
- реакторы (VLR-[...]-Reactor) — [...] здесь заменяет обозначение типа реактора, такие как: DWG, Object и т. д.;
- тип данных (VL-CATCH-ALL-APPLY-ERROR) — специальный объект, содержащий сведения об ошибках, пойманных функцией `vl-catch-all-apply`, и некоторые иные типы.

Примеры возникновения данных разных типов:

```
Command: (type (entlast))
ENAME
Command: (setq ss (ssget))
Command: (type ss)
PICKSET
Command: (type type)
SUBR
```

```
Command: (type dos_version)
EXRXSUBR
Command: (type pi)
REAL
Command: (type 123)
INT
Command: (type "123")
STR
Command: (type (read "123"))
INT
Command: (type 'ss)
SYM
Command: (type (list 1 2 3))
LIST
Command: (setq f (open (findfile "acad.exe") "r"))
#<file "C:\\ACAD\\2002\\acad.exe">
Command: (type f)
FILE
Command: (setq f (close f))
nil
Command: (type nil)
nil
Command: (type 'nil)
nil
Command: (type (vlax-get-acad-object))
VLA-OBJECT
```

Проверка типов очень часто производится в сравнении с `nil`. Любая переменная как бы является логической — ее значение или `nil`, или любое другое. В языке Pascal, например, невозможны выражения наподобие:

```
Var
   MyInt    : integer;
   MyString : string;
Begin
If MyInt then MyString :='Есть число' else MyString :='Нет числа';
end;
```

В языке LISP аналогичное выражение является обычным:

```
(if MyInt (setq MyString "Есть число") (setq MyString "Нет числа"))
```

Поклонников типизированных языков такие возможности приводят в ужас. Справедливо, надо сказать. Строгая типизация была придумана для предотвращения ошибок еще на этапе трансляции программы в машинный код. LISP создавался как интерпретатор и ограничения на типы данных не были предусмотрены сознательно.

Однако даже в бестиповом языке контроль за типами данных необходим. Если передать в функцию данные не того типа, который требуется, будут происходить ошибки. Для контроля за типом данных приходится разрабатывать собственные функции. Подобные функции называются *предикатными* и обычно оканчиваются

символом "p", но мы, для удобочитаемости, будем относить их к группе "ru-is". Примеры некоторых подобных функций приведены в листингах 10.22—10.24.

Листинг 10.22. Функция ru-is-string

```
(defun ru-is-string (value)
;;;  (ru-is-string  "Строка") > T
;;;  (ru-is-string  T) >nil
  (= (type value) 'STR)
)
```

Листинг 10.23. Функция ru-is-real

```
(defun ru-is-real (value)
;;;  (ru-is-real  123) > nil
;;;  (ru-is-real  123.0) > T
  (= (type value) 'REAL)
)
```

Листинг 10.24. Функция ru-is-int

```
(defun ru-is-int (value)
;;;  (ru-is-int  123) > T
;;;  (ru-is-int  123.0) > nil
  (= (type value) 'INT)
)
```

Как видите, они очень просты и вводятся только по рассмотренным ранее причинам.

Кроме контроля типов нужны и преобразования типов данных (листинги 10.25 и 10.26). Стандартные функции вроде atoi, itoa, rtos не требуют пояснений, но их недостаточно.

Листинг 10.25. Функция ru-conv-deg-to-rad

```
(defun ru-conv-deg-to-rad (deg_ang)
  (* pi (/ deg_ang 180.0))
)
```

Листинг 10.26. Функция ru-conv-rad-to-deg

```
 (defun ru-conv-rad-to-deg (rad_ang)
  (* 180.0 (/ rad_ang pi))
)
```

Без предыдущих двух функций (обычно именуемых DTR и RTD) не обходится ни одна программа. Следующие функции осуществляют преобразования между строковыми и логическими данными. Они (листинги 10.27—10.31) пригодятся при чтении данных из текстовых файлов.

Листинг 10.27. Функция ru-conv-str-to-bool

```
(defun ru-conv-str-to-bool (s)
(not (null (member (strcase s)
              '("1" "Y" "" "YES" "ДА" "Д" "T" "TRUE" ".T.")
            );_ end of member
     )
)
);_ end of defun
```

Листинг 10.28. Функция ru-conv-bool-to-str

```
(defun ru-conv-bool-to-str (val)
(if val "1" "0")
)
```

Листинг 10.29. Функция ru-conv-value-to-bool

```
(defun ru-conv-value-to-bool (value / type_val)
;; преобразует значение любого типа в T или NIL
  (cond
    ((= (setq type_val (type value)) 'INT)(/= value 0))
    ((= type_val 'REAL) (/= value 0.0))
    ((= type_val 'STR) (ru-conv-str-to-bool value))
    ((= type_val 'variant)
      (= (vlax-variant-value value) :vlax-true)
    )
     (T  value)
 )
)
```

Листинг 10.30. Функция ru-conv-value-to-wordbool

```
(defun ru-conv-value-to-wordbool (value / type_val)
;;; преобразует значение любого типа в 1 или 0
;;; это требуется для совместимости с COM-серверами
  (cond
    ((= (setq type_val (type value)) 'INT) value)
    ((= type_val 'REAL) (if (/= value 0.0) 1 0))
    ((= type_val 'STR) (if (ru-conv-str-to-bool value) 1 0)
    ((= type_val 'variant)
      (if (= (vlax-variant-value value) :vlax-true) 1 0))
    (T (if value 1 0))
 )
)
```

Листинг 10.31. Функция ru-conv-3dPoint-to-2dPoint

```
(defun ru-conv-3dPoint-to-2dPoint (3d_pnt)
  (list (float (car 3d_pnt)) (float (cadr 3d_pnt)))
)
```

Особое внимание нужно обращать на тип данных при работе с ActiveX. Функции, использующие объектную модель, обрабатывают и возвращают VLA-объекты. Преобразование между "традиционными" типами и VLA-объектами производится с помощью соответствующих встроенных функций. При работе с внешними COM-серверами, написанными на других языках, главным препятствием может оказаться несовместимость типов данных. Эти вопросы мы будем рассматривать при разработке собственных серверов.

Контроль типов данных необходим там, где данные разных типов могут быть введены пользователем или получены из внешнего источника, например, из файла или от другой программы. Есть случаи, когда очень велика вероятность нашей собственной ошибки. Например, приведенная функция `ru-conv-value-to-wordbool` введена из-за того, что логические значения мы передаем обычно в виде `T` или `nil`, а для некоторых COM-серверов логические значения следует передавать в виде 1 или 0. Чтобы не вспоминать каждый раз, какой именно тип нужно передавать в конкретном случае, мы можем всегда при работе с COM использовать эту функцию.

В обычных условиях программист сам должен беспокоиться о соответствии типов данных. Можно, конечно, в каждой функции начать маниакально проверять правильность типов переданных параметров, но тогда система будет расходовать слишком много ресурсов на ненужную деятельность.

Глобальные и локальные переменные

Все переменные в LISP делятся на два вида: *глобальные* и *локальные*. Глобальные переменные постоянно находятся в оперативной памяти и доступны из любой функции, загруженной в тот же сеанс. Глобальные переменные создаются автоматически при присваивании им значения вне тела функции или внутри функции, если переменная с таким именем не объявлена локальной.

Локальные переменные явно описываются в заголовке функции `defun` и видны только внутри этой функции. В начале работы функции локальные переменные имеют значение `nil` (если не было выполнено явное присваивание значений), а по окончании работы этой функции они автоматически удаляются из памяти. Аргументы функций также являются локальными переменными. В начале работы функции аргументы принимают значения, переданные функции при ее вызове.

Примечание

Для разделения двух типов локальных переменных удобно применять наименование "рабочая переменная" для тех локальных переменных, которые не являются аргументами. *(Николай Полещук)*

Использование глобальных переменных *по возможности нежелательно*. Они занимают память, вызывают необходимость отслеживания имен переменных (а локальных переменных с одинаковыми именами в разных пользовательских функциях может быть сколько угодно).

В то же время в самой системе AutoCAD и в поставляемых с ней программах имеется множество глобальных переменных. Убедиться в этом можно, загрузив файл acadinfo.lsp и выполнив команду ACADINFO. Просмотр результатов работы этой программы в файле acadinfo.txt даст вам возможность ознакомиться со множеством

"интимных" подробностей текущего сеанса. Глобальные переменные описаны в нем в виде:

```
(setq AC1_10 20)
(setq AC1_100 26)
(setq AC1_128IN_1FT 1)
```

Глобальных переменных самой системы AutoCAD и библиотеки Express Tools мы насчитали более 600 штук, да еще есть системные переменные, которые по сути также являются глобальными.

Очевидно, что совсем без глобальных переменных, по крайней мере в LISP, не обойтись. Менее очевидно, что все-таки можно предусмотреть средства для запоминания переменной (в файле или в словаре) и последующего восстановления в нужный момент сохраненного значения. Мы будем пользоваться сохранением значений переменных и в файлах, и в словарях. Но нужно трезво оценивать реальные последствия введения глобальных переменных или сохранения/восстановления значений внутри функций. Отрицательные последствия от введения глобальной переменной (занятая память, возможность совпадения имен) могут быть гораздо меньшими, чем подключения функций сохранения и восстановления значений. Если мы введем глобальные переменные (с довольно уникальными именами)

```
(setq *RU_DWG_SCALE* 100) (setq *RU_DWG_UNITS* "мм")
```

используемые почти в каждой функции, это будет лучше, чем чтение их несколько раз в секунду из словаря через целую цепочку функций.

Разумеется, совершенно недопустима ситуация, когда программист просто ленится проверить функцию на наличие специально или нечаянно образовавшихся глобальных переменных.

"Выловить" глобальные переменные просто — выделите текст функции и щелкните кнопку **Check Selection** (Проверить выделенный фрагмент) или **Check Edit Window** (Проверить текст в редакторе). Visual LISP проверит исходный текст и выведет в окно **Build Output** (Сообщения сборки) результаты проверки:

Фрагмент результатов проверки

```
; === Top statistic:
; Function definition (with number of arguments): ((RU-APP-LOAD-ARX . 1))
; === Top statistic:
; Global variables: (*RU_DEVELOPER* RESULT)
; Function definition (with number of arguments): ((_RU-APP-LOAD-LSP . 2) (RU-APP-
LOAD . 1))
```

В данном случае мы видим, в функции `RU-APP-LOAD-LSP` две глобальных переменных. Одна из них `*RU_DEVELOPER*` введена нами сознательно, а `RESULT` мы просто забыли включить в список локальных переменных.

Замечание

Для отображения результатов проверки нужно в диалоговом окне **General Options** (Общие параметры) на вкладке **Diagnostic** (Диагностические) установить флажок **Report statistics during syntax checking** (Вывод статистики в ходе синтаксической проверки).

Откуда берется мусор и как с ним бороться

В результате различных вычислений в памяти компьютера возникают структуры, на которые потом нельзя сослаться. Если результат вычислений не связан с именем какой-то переменной, он отобразится на экране, но его никак нельзя будет использовать.

Пример 1.

```
Command: (list "Этот список будет мусором")
("Этот список будет мусором")
```

На этот список нельзя сослаться, ячейки памяти, занятые им, используются бесполезно.

Пример 2.

```
Command: (setq list_1 '("Это станет мусором" "Остаток списка"))
("Это станет мусором" "Остаток списка")
Command: (setq list_1 (cdr list_1))
("Остаток списка")
```

В данном примере мы присвоили переменной `list_1` новое значение — указатель на "хвост" первоначального значения. "Голова" прежнего значения осталась в памяти, т. к. она хранилась в отдельных ячейках, но сослаться на нее никак нельзя.

LISP с самого начала имел механизм сборки мусора в виде функции `gc` — (Garbage Collector, сборщик мусора), реализованный в конкретных системах, в том числе и в AutoCAD[1]. Сборщик мусора автоматически запускается, когда в памяти остается мало места, перебирает все ячейки и собирает являющиеся мусором ячейки памяти в список свободной памяти. О работе мусорщика можно только догадываться по периодическим задержкам работы. Оценить величину задержек можно по сообщению функции `mem`:

```
Command: (mem)
; GC calls: 27; GC run time: 1790 ms
Dynamic memory segments statistic:
PgSz  Used  Free  FMCL  Segs  Type
 512    49   205    99     2  lisp stacks
 256  2889  2211   209    20  bytecode area
4096   545    10    10    37  CONS memory
  32  1064   919   867     1  ::new
4096   141    99    11    16  DM Str
 512     1   126   126     1  undo strings
4096   390    15    14    27  DMxx memory
 128     4   507   506     1  bstack body
Segment size: 65536, total used: 105, free: 2
```

Сборщик мусора можно вызвать самим, и мы вставим такой вызов в подходящее место. Разумеется, это будет какая-то одна функция.

---

[1] Отслеживание за удалением объектов из памяти после их использования является источником головной боли для программистов на C++. В среде .NET Framework наконец-то появился подобный механизм автоматической сборки мусора.

Ошибки вычислений

Даже при правильном алгоритме в вычислениях могут допускаться ошибки. Некоторые из них зависят от особенности операций с вещественными числами[1]. Точность вычислений вещественных чисел ограничена. При тригонометрических вычислениях это может иметь большое значение.

Считается, что константа "пи" в системе AutoCAD имеет низкую точность:

```
Command: !pi
3.14159
```

При повышенных требованиях к точности эту константу пытаются заменить на более точное значение. На самом же деле низкую точность имеет только выводимое на экран значение `pi`. Проведем эксперимент:

```
Command: (= 3.141592653589793 pi)
T
Command: (= 3.141592653589792 pi)
nil
```

Как видим, сравнение производится с точностью не менее чем до 15-го знака. На самом деле можно получить еще 2 знака, т. е. 3.14159265358979309, но функции сравнения на них уже не реагируют.

Лучше не сравнивать результат тригонометрической функции с константой, иначе возникают ошибки, которые трудно заметить:

```
Command: (setq x (sin 0.0))
0.0
Command: (setq y (sin (* 2.0 pi)))
-2.44921e-016
Command: (= x y)
nil
```

Язык LISP "решил", что sin(0) не равен sin(2PI). Однако! Впрочем, так же мог "решить" и другой язык программирования, но именно в LISP можно получать верный результат путем сравнения с заданной точностью:

```
Command: (equal x y 0.00000001)
T
Command: (equal x y 0.000000000000001)
T
Command: (equal x y 0.0000000000000001)
nil
```

Что применять: =, *eq* или *equal*?

Для проверки значений на равенство возможно использование трех функций. Обычно используют функцию `=`, как более понятную по смыслу. Некоторые про-

[1] Практика показывает, что даже профессиональные программисты, особенно начинавшие сразу с визуальных средств разработки, иногда не понимают особенностей внутреннего представления вещественных чисел в современных вычислительных системах и часто удивляются "непонятному" поведению программ, содержащих дробные вычисления. К сожалению, мы, как и авторы многих современных книг, не можем уделить этому чрезвычайно важному вопросу подобающего ему места.

граммисты (например, автор библиотеки STDLIB Рейни Урбан) используют исключительно функцию `eq`, а некоторые — и `=`, и `eq`, и `equal` без видимой системы. Однако и в простых операциях сравнения возможны ошибки. Пример ошибки при сравнении вещественных чисел мы только что рассмотрели.

Функция `=` определяет, равны ли несколько *атомов*, функция `eq` определяет, идентичны ли два *выражения*, функция `equal` определяет, равны ли *значения* двух выражений с допуском. Если два списка `eq`, они всегда `equal`, но если два списка `equal`, они не обязательно `eq`.

Проверка на практике:

- `(= (list 1 2 3) (list 1 2 3))` вернет `nil` (сравниваются не атомы).
- `(eq (list 1 2 3) (list 1 2 3))` вернет `nil` (выражения не идентичны, это два разных списка, находящихся в разных ячейках памяти, хотя значения их равны).
- `(equal (list 1 2 3) (list 1 2 3))` вернет `T` (значения выражений идентичны).
- `(setq a (list 1 2 3)) (setq b (list 1 2 3)) (= a b)` вернет `nil` (сравниваются не атомы).
- `(eq a b)` вернет `nil` (выражения не идентичны, это два разных списка, находящихся в разных ячейках памяти, хотя значения их идентичны).
- `(equal a b)` вернет `T` (значения выражений идентичны).
- `(setq c a)`.
- `(= a c)` вернет `T` (атомы равны).
- `(eq a c)` вернет `T` (выражения идентичны).
- `(equal a c)` вернет `T` (значения выражений идентичны).
- `(= b c)` вернет `nil`.
- `(eq b c)` вернет `nil` (выражения не идентичны).
- `(equal b c)` вернет `T` (значения выражений идентичны).
- и `(= (* 2 2) 4)`, и `(eq (* 2 2) 4)`, и `(equal (* 2 2) 4)` вернут `T`.
- и `(= (* 2.0 2.0 1.0) (+ 2 2))`, и `(eq (* 2.0 2.0 1.0) (+ 2 2))`, и `(equal (* 2.0 2.0 1.0) (+ 2 2))` вернут `T`.
- `(setq x (* 2 2))`.
- и `(= x 4)`, и `(eq x 4)`, и `(equal x 4)` вернут `T`.

Если эти эксперименты вас окончательно запутали, примите наши рекомендации:

- используйте функцию `=` для сравнения целых чисел и строк;
- используйте функцию `eq` для проверки, не указывают ли две переменных на один и тот же список (адрес в памяти);
- используйте функцию `equal` для проверки равенства значений выражений (для чисел и списков с возможностью допуска) — обычно именно это и требуется.
- функция `=` не всегда вернет `T` при равенстве значений.

Если возникают подозрения, что проверка производится неправильно, заменяйте функцию `=` на функцию `equal`. Осторожней пользуйтесь рекомендациями, заимство-

ванными из описаний других LISP-систем — в них могут быть другие принципы, например, `(equal 4.00 4)` может возвращать `nil`, хотя Visual LISP вернет `T`.

NOT и NULL

Функция-предикат `(not выражение)` возвращает `T`, если выражение — `nil`, иначе возвращает `nil`. Функция-предикат `(null выражение)` возвращает `T`, если выражение *вычисляется* в `nil`, иначе возвращает `nil`. Разница небольшая и результат одинаковый. Обычно `null` используется для проверки на пустой список, а `not` больше подходит по смыслу для проверки логических условий. Две функции были введены для лучшего выражения смысла. Например:

```
(if (not (null list_params);; если не пустой список
(do list_params)          ;; обработать список
)
```

Сравните с другим выражением:

```
(if (not test params);; если не выполнен тест
  (exit)              ;; выйти
  (do params)         ;; иначе обработать параметры
)
```

Применяйте `not` или `null` так, чтобы лучше выражался смысл.

Ошибки вызова команд

"Как я уже говорил, я не люблю повторяться по два раза", но вы не забыли, что имена команд должны начинаться с префикса "_.", опций — с префикса "_"? Тогда идем дальше.

Часто "рисование" с помощью функции `command` происходит неправильно из-за того, что программист плохо изучил особенности команды и все возможные варианты ее работы. Он самоуверенно считает, что если у него на компьютере все работает, то так будет всегда. Вот, самый типичный пример.

Неправильный вызов команды TEXT

Допустим, мы написали функцию:

```
(defun my_txt (txt / pt1)
 (initget 1)
 (setq pt1 (getpoint "\nНачало текста: "))
 (command "_.TEXT" pt1 "2.5" 0 txt)
)
```

У нас всегда в текстовом стиле была установлена высота 0, и *всегда* все работало. "А эти бестолковые пользователи установили фиксированную высоту текста, и начались сбои (2.5 воспринимается как угол, 0 — как текст и дается лишний ответ, воспринимаемый как следующая команда). Ну, тупые!" — думает автор.

Возможен и противоположный вариант. Функция пишется под фиксированную высоту текста:

```
(defun my_txt (txt / pt1)
 (initget 1)
 (setq pt1 (getpoint "\nНачало текста: "))
 (command "_.TEXT" pt1 0 txt)
)
```

Теперь сбой произойдет, но только если у пользователя текущий текстовый стиль имеет нулевую высоту. Ну, эти еще тупее!

Но делать надо было так, как показано в листинге 10.32.

Листинг 10.32. Функция `true_text`

```
(defun true_text (txt pt1 txt_height txt_angle txt_just / old_osnap)
;;; (tru_text текст точка_начала высота_текста
;;;  угол_поворота опция_выравнивания)

  (setq old_osnap (getvar "OSMODE"))
  (setvar "OSMODE" 0)
  (if (= (cdr (assoc 40 (tblsearch "STYLE" (getvar "TEXTSTYLE")))) 0.0)
  ;; нулевая высота текста
      (if txt_just
        (command "_.TEXT" "_J" txt_just pt1 txt_heigh txt_angle txt)
        (command "_.TEXT" pt1 txt_heigh txt_angle txt)
     )
    ;; фиксированная высота
      (if txt_just
         (command "_.TEXT" "_J" txt_just pt1 txt_angle txt)
        (command "_.TEXT" pt1 txt_angle txt)
     )
 )
  (setvar "OSMODE" old_osnap)
)
```

Вот теперь можно забыть команду TEXT и программно писать через функцию, например:

```
(defun my_txt (txt / pt1)
 (initget 1)
 (setq pt1 (getpoint "\nНачало текста: "))
 (true_text txt pt1 2.5 0 nil)
)
```

Замечание

Это был пример функции, написанной в "командном" стиле. Написав "надо было делать так", мы имели в виду только вариант с использованием `command`. Хотя функция `true_text` рабочая, в своей системе тексты писать мы будем объектными методами.

Забываем отключить объектную привязку

Второй самый распространенный пример связан с тем, что неверное рисование происходит из-за включенного режима привязки. Бестолковый пользователь, видите

ли, не знает, что при работе с нашими "супер-пупер" программами объектную привязку надо отключать!

Закомментируем в предыдущем примере некоторые строки.

```
(defun true_text (txt pt1 txt_height txt_angle txt_just / old_osnap)
;;;  (setq old_osnap (getvar "OSMODE"))
;;;  (setvar "OSMODE" 0)
  (if (= (cdr (assoc 40 (tblsearch "STYLE" (getvar "TEXTSTYLE")))) 0.0)
  ;; нулевая высота текста
      (if txt_just
        (command "_.TEXT" "_J" txt_just pt1 txt_heigh txt_angle txt)
        (command "_.TEXT" pt1 txt_heigh txt_angle txt)
     )
    ;; фиксированная высота
      (if txt_just
         (command "_.TEXT" "_J" txt_just pt1 txt_angle txt)
        (command "_.TEXT" pt1 txt_angle txt)
     )
 )
;;;  (setvar "OSMODE" old_osnap)
)
```

Теперь, если у пользователя включен какой-либо постоянный режим объектной привязки, рисование может оказаться непредсказуемым.

Намек! Совет!! Требование!!!

Перед любым программным рисованием *с использованием функции* **command** программно отключайте объектную привязку, а после рисования восстанавливайте режим!

Для облегчения действий по отключению объектной привязки можно написать две полезные функции (листинги 10.33 и 10.34).

Листинг 10.33. Функция `ru-draw-no-osnap`

```
(defun ru-draw-no-osnap ()
  (setq *RU_OLD_OSNAP* (getvar "OSMODE"));;глобальная переменная
  (setvar "OSMODE" 0)
)
```

Листинг 10.34. Функция `ru-draw-old-osnap`

```
(defun ru-draw-old-osnap ()
  (if *RU_OLD_OSNAP*
    (setvar "OSMODE" *RU_OLD_OSNAP*)
 )
)
```

В любой программе теперь после всех операций ввода, во время которых пользователь должен иметь возможность воспользоваться хоть постоянной, хоть временной привязкой, пишем:

```
(ru-draw-no-osnap)
(command "_.LINE" pt1 pt2 pt3 "")
(command "_.LINE" pt4 pt5 "")
(ru-draw-old-osnap)
```

А почему же мы не использовали эти функции в листинге 10.32? Да потому, что функция `true_text` полностью включает все операции подготовки, отрисовки и завершения при создании текста и мы можем обойтись локальными переменными. При разработке конечных прикладных программ бывают сложные ситуации, с вызовом множества "рисовальных" функций (возможно чужих) и лучше перестраховаться, чтобы гарантированно не возникало "загадочных" рисунков. Все свои функции рисования мы постараемся написать так, чтобы состояние объектной привязки не влияло на результат, но ошибки делают все и лишняя пара "страховочных" функций нам не повредит.

Структура программы

Хорошая программа должна легко читаться, а для этого ее текст должен быть хорошо структурирован. В труднодостижимом идеале вся программа должна помещаться на один экран. Для больших программ это кажется невозможным, но это только кажется. Надо только руки приложить.

Дать однозначно верных рекомендаций по структуре программ нельзя. Все зависит от назначения и сложности программы. Рекомендованный размер видимой части программы условен. Встречаются программы из 20 строк, понять логику работы которых можно с большим трудом, но бывают и огромные программы, в которых все понятно.

Замечание

В качестве примера можно привести очень хорошую программу ACADVar[1] Алексея Васильченко из Таганрога. Помимо высоких потребительских качеств (отличная инсталляция, справочная система, тщательно отработанные диалоговые окна) эту программу отличает великолепный код. Размер LSP-файла программы 250 Кбайт, в нем определены более 100 функций, но все очень понятно благодаря продуманной системе имен функций и переменных, подробным комментариям и четкой структуризации. Ясность мыслей автора подкреплена ясностью их изложения. Вся "главная" программа состоит из вызова единственной функции. Суперминимализм!

Варианты структуры программы

Попытаемся выработать заготовку структуры для LISP-программы на уровне отдельного файла. Напишем условную, но работающую программу. Условность программы подчеркнем нарушением всех наших правил по именам функций и переменных.

---

[1] Скачать программу бесплатно можно с сайта Геннадия Поспелова **cadhlp.da.ru**. На этом сайте можно найти множество бесплатных программ, документов, шрифтов и прочих замечательных вещей.

Вариант "один вход — один выход"

Листинг 10.35. Первый вариант структуры

```
(defun ПРОГРАММА (ПАРАМЕТРЫ)
(НАЧАЛО)
  (if (ПАРАМЕТРЫ_ПРОВЕРЕНЫ)
    (progn
      (ЧТЕНИЕ_НАСТРОЕК)
      (if (ДАННЫЕ_ВВЕДЕНЫ)
       (РАБОТА)
       (СООБЩЕНИЕ_ОБ_ОШИБКЕ)
     )
      (СОХРАНЕНИЕ_НАСТРОЕК)
   )
    (СООБЩЕНИЕ_ОБ_ОШИБКЕ)
 )
  (КОНЕЦ)
  (princ)
)
```

Мы разбили свою программу на укрупненные блоки, каждый из которых можно описать в виде функции. Сейчас мы специально присвоили таким функциям-блокам русские имена. Кстати, работать программа будет и с русскими именами функций. Некоторые блоки (НАЧАЛО, КОНЕЦ, СООБЩЕНИЕ_ОБ_ОШИБКЕ) могут быть стандартными для всех программ, некоторые могут быть описаны в файле программы. Мы уже говорили о том, что у нас во всех программах основная функция будет иметь имя START. При загрузке каждого файла функция START переопределяется. Точно так же может переопределяться функция РАБОТА или вместо нее может быть вызвана какая-то библиотечная функция.

В приведенной схеме ПРОГРАММА имеет один вход и один выход. Даже если происходят сбойные ситуации, при которых функции ПАРАМЕТРЫ_ПРОВЕРЕНЫ и ДАННЫЕ_ВВЕДЕНЫ вернут nil, все равно произойдет штатный выход. Структура программы предполагает, что все используемые функции определены вне тела функции ПРОГРАММА. Возможно размещение определений этих функций внутри тела главной функции, в этом случае их имена следует внести в список локальных переменных. Конечно же, это могут быть и не функции, а просто логический блок. Не стоит выделять в отдельные функции простые последовательности наподобие ввода пары точек и нескольких элементарных вычислений. Определения функций-блоков мы рассмотрим далее, пока же сосредоточимся на общей структуре программы.

Вариант "один вход — несколько выходов"

Встречается и другая структура программы, в которой имеется один вход, но несколько выходов. Если не выполняется какая-то проверка, то производится выход из программы. Такая структура нагляднее, но имеет существенный недостаток — много запасных выходов труднее контролировать.

Листинг 10.36. Второй вариант структуры

```
(defun ПРОГРАММА (ПАРАМЕТРЫ)
 (НАЧАЛО)
  (if (not (ПАРАМЕТРЫ_ПРОВЕРЕНЫ)) (ДОСРОЧНЫЙ_ВЫХОД))
  (ЧТЕНИЕ_НАСТРОЕК)
  (if (not (ДАННЫЕ_ВВЕДЕНЫ))(ДОСРОЧНЫЙ_ВЫХОД))
  (РАБОТА)
  (СОХРАНЕНИЕ_НАСТРОЕК)
  (КОНЕЦ)
  (princ)
)
```

Хорошая программа не должна оказывать своей работой влияния на общее состояние системы AutoCAD. Это означает, что все настройки, хранящиеся в системных переменных, после работы программы должны остаться такими же и после завершения любой программы. Достигается это за счет использования функций НАЧАЛО и КОНЕЦ.

В функции НАЧАЛО должно быть вложено переопределение стандартной функции обработчика ошибок *error*, сохранение значений системных переменных, установка их новых, требуемых для программы, значений. В функции КОНЕЦ предусматривается восстановление значений системных переменных и стандартной функции *error*.

При нештатном выходе необходимо также восстановить все параметры. Поэтому в примере мы указали не функцию exit (о ней *см. далее*), а некую функцию ДОСРОЧНЫЙ_ВЫХОД, которая должна известить об ошибке, восстановить окружение и выйти из программы (листинг 10.37).

Листинг 10.37. Прототип функции ДОСРОЧНЫЙ_ВЫХОД

```
(defun ДОСРОЧНЫЙ_ВЫХОД ()
    (СООБЩЕНИЕ_ОБ_ОШИБКЕ)
    (КОНЕЦ)
    (exit)
    (princ)
 )
```

Вот тут и кроется ловушка. При досрочном выходе с помощью функции exit срабатывает функция *error*. Мы ее специально переопределяли (заменяли сообщение об ошибке, возможно, еще что-то предусматривали, но вынуждены восстанавливать стандартный обработчик ошибок до вызова функции exit. Мы не сможем воспользоваться собственным обработчиком ошибок именно тогда, когда он нам нужен. В результате некоторое улучшение читабельности программы ухудшает ее потребительские свойства. А ведь в нашей программе было всего два нештатных выхода, в реальных условиях их может быть больше. Так что вы как хотите, а мы в своих программах будем придерживаться первого варианта структуры.

Приведем полностью текст программы с функциями и переменными (листинг 10.38). Параметры мы ввели бессмысленные, просто для демонстрации блока проверки.

Листинг 10.38. Полный текст примера структуры

```
(defun ПРОГРАММА (ПАРАМЕТРЫ / ИЗВЕЩЕНИЕ РАЗ ПЕСНЯ РАБОТА
 СООБЩЕНИЕ_ОБ_ОШИБКЕ ДАННЫЕ_ВВЕДЕНЫ СОХРАНЕНИЕ_НАСТРОЕК ЧТЕНИЕ_НАСТРОЕК
 ПАРАМЕТРЫ_ПРОВЕРЕНЫ КОНЕЦ НАЧАЛО)
;;;#### Локальная функция ####
  (defun НАЧАЛО ()
    (princ "\nЩАС КАК ЗАПОЮ!")
    (setq ИЗВЕЩЕНИЕ "")
)
;;;#### Локальная функция ####
  (defun КОНЕЦ ()
    (princ "\nПЕСНЯ КОНЧИЛАСЬ! ЩАС ЕЩЕ КАК ЗАТАНЦУЮ...")
 )
;;;#### Локальная функция ####
  (defun ПАРАМЕТРЫ_ПРОВЕРЕНЫ ()
    (if ПАРАМЕТРЫ
       T
      (progn
        (setq ИЗВЕЩЕНИЕ (strcat ИЗВЕЩЕНИЕ "\nНе задан параметр"))
        nil
      )
   )
 )
;;;#### Локальная функция ####
  (defun ЧТЕНИЕ_НАСТРОЕК ()
    (setq РАЗ (getvar "USERI1"))
 )
;;;#### Локальная функция ####
  (defun СОХРАНЕНИЕ_НАСТРОЕК ()
    (setvar "USERI1" РАЗ)
 )
;;;#### Локальная функция ####
  (defun ДАННЫЕ_ВВЕДЕНЫ ()
    (setq РАЗ (getint "\nСколько раз: "))
    (cond
      ((> РАЗ 0)
       РАЗ
     )
      (T
       (setq ИЗВЕЩЕНИЕ (strcat ИЗВЕЩЕНИЕ "\nМаловато РАЗ будет"))
       nil
      )
   )
 )
;;;#### Локальная функция ####
  (defun СООБЩЕНИЕ_ОБ_ОШИБКЕ     ()
    (alert (strcat "ОШИБКИ: \n" ИЗВЕЩЕНИЕ))
 )
```

```
;;;#### Локальная функция ####
  (defun РАБОТА (/ КОТОРЫЙ_РАЗ)
    (setq КОТОРЫЙ_РАЗ 1)
    (setq ПЕСНЯ "")
    (repeat РАЗ
      (cond
        ((= КОТОРЫЙ_РАЗ 1)(setq ПЕСНЯ (strcat ПЕСНЯ "\nЭх, раз")))
        ((= КОТОРЫЙ_РАЗ 2)(setq ПЕСНЯ (strcat ПЕСНЯ "\nДа еще раз")))
        ((= КОТОРЫЙ_РАЗ 3)(setq ПЕСНЯ (strcat ПЕСНЯ "\nДа еще много")))
        (t (setq ПЕСНЯ (strcat ПЕСНЯ "\nмного")))
     )
      (setq КОТОРЫЙ_РАЗ (1+ КОТОРЫЙ_РАЗ))
   )
    (if (> КОТОРЫЙ_РАЗ 2)
      (setq ПЕСНЯ (strcat ПЕСНЯ " раз!"))
      (setq ПЕСНЯ (strcat ПЕСНЯ "!"))
   )
    (princ ПЕСНЯ)
 )
;;;#### ГЛАВНАЯ ФУНКЦИЯ ####
  (НАЧАЛО)
  (if (ПАРАМЕТРЫ_ПРОВЕРЕНЫ)
    (progn
      (ЧТЕНИЕ_НАСТРОЕК)
      (if (ДАННЫЕ_ВВЕДЕНЫ) (РАБОТА) (СООБЩЕНИЕ_ОБ_ОШИБКЕ))
      (СОХРАНЕНИЕ_НАСТРОЕК)
   )
    (СООБЩЕНИЕ_ОБ_ОШИБКЕ)
 )
  (КОНЕЦ)
  (princ)
)
```

Дополнительные пояснения:

- локальные функции объявлены внутри главной и имена локальных функций внесены в список локальных переменных;
- переменные, локальные внутри главной функции, видны внутри локальных функций;
- мы не передаем переменные в виде параметров в локальные функции, чтобы неоправданно не усложнять программу;
- внутри функции РАБОТА объявлена локальная переменная, видимая только внутри этой функции;
- мы предусмотрели добавление в строку ИЗВЕЩЕНИЕ сообщений об ошибке из разных функций, хотя реально пройдет только одно, чтобы показать, что можно "аккумулировать" много сообщений, позволяющих известить обо всех ошибках сразу, не заставляя много раз запускать программу.

Программы, использующие диалоговые окна

Программы, использующие диалоговые окна, работают по более сложной схеме. Диалоговые окна чаще всего используются для ввода информации. Контроль за правильностью ввода данных должен быть "зашит" в блок работы с диалоговым окном, чтобы после перехода к блоку обработки данных быть уверенными, что все данные корректны. Блок работы с диалогом также нуждается в четкой структуризации. Во время работы программы может потребоваться скрытие диалогового окна, ввод данных из рисунка, вывод других диалоговых окон.

Использование диалоговых окон

Диалоговые окна в системе AutoCAD нужно применять разумно. Когда они только появились, начался массовый перевод программ в диалоговый режим. Потом выяснилось, что далеко не всегда это нужно делать. Командная строка, которую мы не раз восхваляли, иногда оказывается гораздо удобнее. Представьте себе, как неудобно было бы рисовать полилинии, если бы команду PLINE (ПЛИНИЯ) перевели в диалоговый режим. А он для этой команды так и напрашивается — множество опций, различные варианты действий. Но поковыряться в таком диалоге интересно раза два, а в повседневной работе он оказался бы тормозом.

Замечание

Далее мы будем говорить не только и не столько про диалоговые окна, создаваемые с использованием языка DCL (Dialog Control Language). Не удивляйтесь, если будут обсуждаться возможности, недоступные в этом языке.

Ненаучная классификация диалоговых окон

Не случайно описание работы с диалогами мы поместили в главу, значительная часть которой посвящена ошибкам программирования. Анализируя практику применения диалоговых окон, можно заметить, что они бывают вредными, бесполезными, умеренно полезными, полезными и незаменимыми. Любой из диалогов можно еще превратить в *опасный*.

Вредные диалоги

К вредным можно отнести диалоги, появляющиеся, когда в них не нуждаются, и отвлекающие пользователей на их закрытие. Типичный пример — диалоговое сообщение о действиях программы. Диалог, наподобие `alert`, нужен только тогда, когда действительно требуется привлечь внимание пользователя. При этом необходимо вывести в диалог максимум полезной информации. Если диалог повторит стандартное сообщение "Function cancelled", пользы от него ни на грош. Сообщение "Файл не найден" неинформативно. Необходимо указать, какой файл не найден, причем с полным именем, только тогда будет польза.

Подобные диалоги могли бы стать *не очень вредными* и даже полезными, если бы пользователь мог воздействовать на их появление. Можно сделать диалог с опцией-переключателем "больше не показывать такие сообщения". После установки переключателя сообщения, если они нужны, могли бы выводиться только в командную строку, а при включении режима отладки — снова в диалоговом окне. Но такой прием требует определенной квалификации программиста.

Замечание

Гораздо проще предусмотреть зависимость необходимости и способа оповещения пользователя о возникновении той или иной ситуации в зависимости от состояния глобальной переменной. По аналогии с действием системной переменной EXPERT. Почему бы не воспользоваться непосредственно этой переменной? Не надо забывать, что пользователь может быть экспертом в системе AutoCAD и полным (пожалуй, правильнее сказать — пустым) "чайником" в нашей системе. В случае использования глобальной переменной, например, `ru-expert-prompt-mode` появление предупреждающих диалоговых окон или использование предупреждения в командной строке, как и сам факт появления предупреждений, может предельно просто регулироваться в любых необходимых пределах. В случае возникновения ситуации, для которой предусмотрено предупреждение, мы сможем проверить состояние этой глобальной переменной и, сравнив ее состояние со степенью важности предупреждения, которое должно быть задано нами при разработке, решить, необходимо ли оповещение, и выбрать его способ. *(Петр Лоскутов)*

Бесполезные диалоги

Бесполезные диалоги являются такими потому, что только закрывают экран, не давая никаких дополнительных возможностей. Пример бесполезного диалога — функция `dos_getstring` библиотеки DOSLib. Ввод из командной строки перенесен в диалоговое окно с возможностью установки значения по умолчанию. Красивые кнопки сначала понравятся ("Ой, Вань, гляди, какие кнопочки!"), но потом будет вызывать раздражение, что появившееся диалоговое окно заслоняет экран. Например, запрашивается позиция, которая будет вставлена после только что нарисованной выноски, нужно визуально оценить расположение текста, а тут приходится двигать диалоговое окно, которое нельзя временно скрыть. Здесь дело не в качестве функции `dos_getstring`, а в ее неправильном применении. Более рациональным мог бы быть запрос в командной строке:

```
Текст позиции изделия <5>:
```

особенно если по умолчанию будет предложено предварительно автоматически вычисленное значение.

В то же время, если программа построена так, что сначала запрашивается ввод строки, а потом точка ее вставки, диалоговый ввод (но более изощренный) мог бы стать полезным.

Умеренно полезные диалоги

Диалоговые окна могут быть умеренно полезными. Допустим, мы разработали функцию, в которой ввод числа производится через диалоговое окно с контролем допустимых значений (минимальное и максимальное). Именно так работает функция `dos_getreal` из библиотеки DOSLib. Вроде бы это очень полезная функция, но при ее использовании мы теряем важное преимущество AutoCAD — возможность ввода чисел указанием расстояния в рисунке. Во многих случаях именно такая возможность, а не наличие красивых кнопок дает больше реальной пользы. Остальное можно предусмотреть и в командной строке. Считать этот диалог умеренно полезным можно потому, что в нем все-таки имеется полезная возможность контроля диапазона данных.

Приведем пример собственного умеренно полезного (а может быть, и бесполезного) диалогового окна, применяющегося в системе BestIA (рис. 10.1).

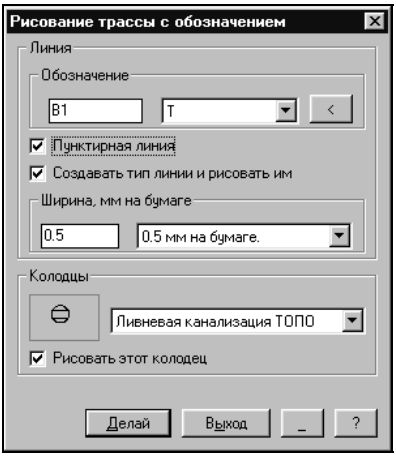


Рис. 10.1. Диалоговое окно рисования трасс

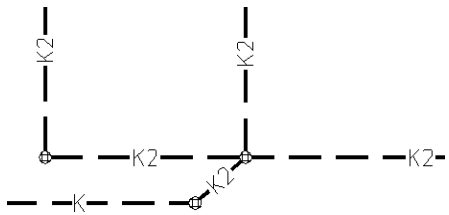


Рис. 10.2. Результат рисования трассы

В этом диалоговом окне можно:

- ввести вручную, выбрать из списка или указать в рисунке буквенно-цифровое обозначение трубопровода;
- задать, требуется ли в дополнение к обозначению рисовать трассу пунктиром;
- указать, будет ли обозначение вставляться в виде текста в разрыв линии, или нужно создать специальный тип линии;
- задать ширину линии на бумаге;
- указать, требуется ли рисовать в узлах трассы колодцы, а если требуется, выбрать тип колодца из списка и увидеть, как он будет выглядеть.

При рисовании заданного изображения (щелчок по кнопке **Делай**) диалоговое окно скрывается, программа создает требуемый тип линии, в цикле, до завершения по опции **Выход** запрашивает пользователя:

```
Начальная точка - Пред/Образец/<Выход>:
Следующая точка - Начало/Образец/Пред/<Выход>:
```

При указании точек программа рисует трассу требуемым типом линии, вставляя, если задано, в узлы пересечений блоки колодцев. При выборе опции **Пред** трасса будет рисоваться от предыдущего узла. При выборе опции **Образец** и указании полилинии-образца трасса будет построена по координатам вершин полилинии с установкой в узлах колодцев (рис. 10.2).

После выхода из интерактивного режима диалоговое окно выводится вновь, в нем можно изменить настройки, временно скрыть окно для обозрения результатов, получить справку. Окончательное завершение работы программы происходит при щелчке по кнопке **Выход**.

Почему же мы столь необходимый диалог относим всего лишь к умеренно полезным, да и то с оговоркой? Да потому, что столь широкие возможности по настройке нужны только очень небольшому количеству пользователей. Большинство из них

делают конкретные разделы проекта и им нужно рисовать, например, только трассы В1, К1 и К2, они не нуждаются во всех типах колодцев и диалог только тормозит их работу. В диалоговом окне 14 элементов управления, пользователь должен задержаться на каждом, чтобы осмыслить его состояние. При этом достаточно велика вероятность ошибки.

Мы предвидели такую ситуацию и предусмотрели вызов функции рисования из иллюстрированного дерева меню. В таком меню пользователь выбирает одним щелчком конкретный вид трассы и быстро его создает. Практика показала, что именно недиалоговым вариантом пользователи чаще и пользуются.

Этот пример мы привели, чтобы предостеречь от излишнего увлечения "хитрыми" диалогами. Программисты часто увлекаются самим процессом решения задач и делают *слишком* хорошо. Усилия, затраченные на разработку многих диалогов, можно направить на более актуальные задачи. Впрочем, некоторым пользователям, для которых счастье достигается не в покое, а в борьбе с неприятностями, работа со сложной программой будет привлекательней, чем с простой.

Полезные диалоги

Диалог полезен тогда, когда предоставляет пользователю дополнительные удобства. Бесполезный диалог ввода строки можно сделать полезным, если предусмотреть передачу значения по умолчанию, флага, разрешающего или запрещающего ввод пустой строки, возможности выбора строк из пополняемого словаря, из списка ранее вводимых значений и способность взять текст из рисунка указанием на любой примитив, внешне похожий на текст. Пример такого диалога приведен в этой главе.

Ввод диаметра трубопровода, о котором мы упоминали, можно значительно усовершенствовать, добавив возможность выбора из иерархического справочника, не исключая при этом ручной ввод. Кому-то проще набрать значение диаметра на клавиатуре, а кто-то предпочтет выбор из справочника, особенно если это не просто диаметр условного прохода, а нужно еще и вспомнить, какой именно должен быть текст — 133×4, 133×3.5 или 133×3.2.

Как делаются *такие* диалоги, мы разберем в последующих главах.

Незаменимые диалоги

Незаменимыми всегда являются диалоговые окна специализированных программ. Часто для таких программ требуется установка множества параметров, ввести которые удобно именно в диалоге. Примером такого диалога может служить любая программа рисования формата. Незаменимыми могут быть диалоги, значительно облегчающие какие-то действия.

Взгляните на диалоговое окно редактирования координат полилинии (рис. 10.3). Такой диалог незаменим для специалистов, рисующих трассы и контуры по заданной ведомости координат. Но этот же диалог оказывается незаменимым для многих проектировщиков. В нем можно не только вводить и редактировать координаты, но можно и просто просмотреть весь контур линии вне ее окружения (указав из программы на полилинию). Это позволяет, указав на видимый участок, увидеть всю линию и оценить ее "правильность" или выявить дефекты без многочисленных панорамирований и масштабирований изображения.

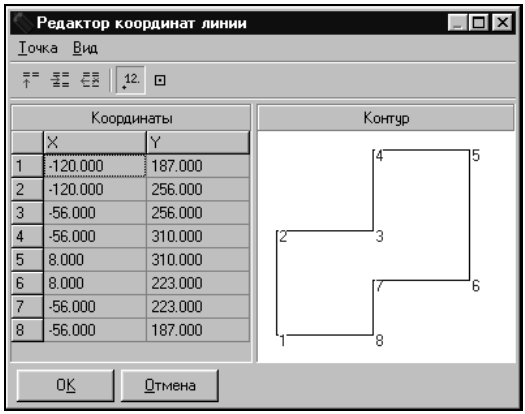


Рис. 10.3. Диалоговое окно **Редактор координат полилинии**

Опасные диалоги

Некоторые диалоги просто опасны. Часто неопытные программисты слишком увлекаются радикальными средствами, доступными в современных средах разработки. В *модальном* диалоговом окне, например, может выполняться редактирование указанного в рисунке текста с синхронным изменением текста на графическом экране. При этом сразу, по мере корректировки, вносятся изменения в графическую базу данных рисунка. Это весьма неразумно. Любой диалог должен иметь возможность отказа от произведенных с его использованием действий. Для того он, диалог, и сделан.

Замечание

Сказанное не относится к *немодальным* диалоговым окнам, таким как инструментальные панели, окно свойств или Центра управления в системе AutoCAD. Модальное диалоговое окно блокирует доступ к остальной части системы до тех пор, пока не будет закрыто.

Немодальные окна постоянно "плавают" на экране, позволяя переключаться между ними и документом. Главный недостаток — перекрытие полезной информации. В системе AutoCAD 2004 этот недостаток ликвидирован (окна могут сворачиваться в вертикальную полосу и в нужный момент снова разворачиваться).

Создать немодальное окно средствами языка DCL нельзя.

Если действительно по логике программы требуется внесение каких-то изменений без закрытия диалогового окна, то должна быть предусмотрена дополнительная кнопка **Применить**. Такая кнопка должна быть доступна, если есть изменения исходных данных. После щелчка по ней название кнопки **Отменить** должно изменяться на **Выход**, т. к. отменить уже ничего нельзя. Кнопка **OK** в таком диалоговом окне может оказаться не нужной, т. к. после использования **Применить** название **OK** должно бы также измениться на **Выход**. Признаком хорошего тона при создании подобных диалогов является запоминание исходного состояния системы до применения изменений и, соответственно, переименование кнопки **Отменить** в **Восстановить**, разумеется, кроме тех случаев, когда восстановление невозможно в принципе или требует неадекватно больших усилий.

Опасны диалоговые окна, у которых кнопкой по умолчанию является кнопка опасного действия. Если, например, выводится диалоговое окно с вопросом: "Вы действительно хотите удалить файл?" и по умолчанию фокус находится на кнопке **Да**, то нетерпеливый пользователь непременно нажмет клавишу <Enter> и удалит файл, хотя и не хотел этого делать.

О выходе из диалогов

Любая программа и диалоговое окно должны иметь ясно обозначенный выход! Лучше всего, если этот выход соответствует используемому в Windows стандарту. В программе это пункт меню **Файл | Выход** (но не пункт **Выход** в правой стороне меню) и комбинация клавиш <Alt>+<F4>. Почти все окна имеют кнопку закрытия окна. Щелчок по ней приводит к закрытию окна.

Примечание

Некоторые программисты, под предлогом борьбы с "излишествами" и для упрощения работы, предусматривают выход из программы только по кнопке закрытия окна. Никакого упрощения работы на самом деле не происходит. Пользователь начинает искать привычный выход, не находит, начинает "шевелить мозгами", "чайник" будет спрашивать у знатоков, "тетка" будет долго прицеливаться в маленькую кнопку. При этом, по идее автора, они еще и должны восхищаться его "крутизной".

Диалоговые окна меню не имеют, но имеют кнопки (фактически являющиеся меню), в том числе кнопку закрытия окна. Щелчок по этой кнопке и нажатие клавиши <Esc> должны закрывать диалоговое окно без каких-либо последствий для основной программы.

Если окно выводится только для информации (например, функцией `alert`), в нем имеется только одна кнопка (обычно **OK**) и такое окно закрывается и щелчком по кнопке, и нажатием клавиши <Esc>. Но если в диалоге предусматриваются какие-то действия, или должны возвращаться разные результаты при нажатии **OK** (или других кнопок) и щелчке по "крестику" или нажатии клавиши <Esc>, то кнопка **Отменить** должна быть в диалоговом окне.

У кнопки должно быть правильное название, соответствующее действию и состоянию системы. Если название **ОТМЕНА**, то что-то обязано отменяться, например все, что введено в диалоге, но еще не применено. Если отменить уже невозможно, то кнопка остатается, но становится недоступной.

Это правило должно соблюдаться неукоснительно. Даже если программист считает, что **Отменить** "используется чрезвычайно редко" и "чтоб не перегружать диалог" кнопку **Отменить** можно убрать, а пользователи при этом вынуждены нажать клавишу <Esc> для выхода без изменений. Да еще уверяет, что так "народ хочет".

Примечание

Встречаются диалоговые окна вообще без кнопок и заголовков. Применение таких окон допустимо только тогда, когда они автоматически появляются и исчезают. Иногда в подобном стиле делают диалоговые окна **About** (О программе). В таких окнах отсутствие кнопок допустимо, но нежелательно. Допустимо потому, что программисты традиционно вставляют в эти диалоговые окна всяческие "приколы", невидимые кнопки, секретные области для собственного удовлетворения и развлечения пользователей. Нежелательно потому, что не все настроены на решение загадок. Подобное окно должно закрываться по щелчку в любом его месте или после нажатия клавиши <Esc>.

О цветовой гамме

При конструировании диалогов средствами языка DCL с цветом элементов, к счастью, не "поиграешь". К счастью потому, что далеко не все разработчики имеют чувство меры и стремятся сделать "красиво", а не элегантно и гармонично. Однако диалоги можно проектировать не только средствами DCL и в этом случае нужно быть внимательным. Никогда не устанавливайте фиксированный цвет для элементов диалоговых окон, даже "точно такой же, как у AutoCAD". Если вы установите для некоторых компонентов статические цвета, то при изменении цветовой схемы Windows окно программы может принять омерзительный вид. Рекомендуем проверять, как выглядят ваши окна при изменении цветовой схемы Windows.

Разработка диалогов с использованием DCL

Язык описания диалоговых окон DCL (Dialog Control Language) безнадежно устарел. Но это ясно теперь, когда разработчики избалованы визуальными средствами разработки приложений, используемыми в Windows, а когда-то графические диалоговые окна в AutoCAD выглядели гостями из будущего. Разрабатывать диалоговые программы для AutoCAD оказалось достаточно просто. Теперь мы видим и ругаем недостатки DCL (плавающий размер полей, отсутствие стандартных для Windows компонентов), но продолжаем их применять. Современные средства разработки позволяют вызывать из Visual LISP и "настоящие" диалоговые окна любой сложности, использующие любые элементы интерфейса Windows (в следующих главах мы узнаем, как это делается). Но диалоговые окна, созданные средствами DCL, нельзя (или очень сложно) заменить, если требуется не только ввод данных в элементах диалога, но и обращение к графическому экрану AutoCAD — указание точек, выбор примитивов, измерение расстояний, промежуточные операции интерактивного рисования.

Пример диалоговой функции

Разберем работу с диалоговым окном на примере функции ввода строки `ru-dlg-dcl-get-string`, которая практически всегда будет применяться вместо стандартной функции `getstring`. Именно такой диалог чуть ранее мы приводили в качестве примера полезного диалога. Вызов функции должен производиться так:

```
(ru-dlg-dcl-get-string заголовок_окна старая_строка запрет_пустой_строки)
```

Диалоговое окно (рис. 10.4) выводится с заданным заголовком `заголовок_окна`. В поле редактирования текста **Ввод** помещается `Старая_строка`. Пользователь может просто отредактировать строку, выбрать ее из раскрывающихся списков **История** и **Словарь**, выбрать указанием в рисунке щелчком по кнопке с изображением <. Поле редактирования можно очистить и вернуть в него прежнее значение. Если установлен флажок **В словарь**, содержимое поля редактирования после выхода по кнопке **OK** будет добавлено в личный словарь пользователя. При выходе по кнопке **OK** возвращается содержимое поля редактирования, при выходе по кнопке **Отмена**, нажатию клавиши <Esc> или при щелчке по кнопке закрытия окна возвращается `nil`.

Если передан `запрет_пустой_строки`, то кнопка **OK** недоступна, пока поле редактирование пустое (рис. 10.5).

Текст DCL-файла приведен в листинге 10.39.

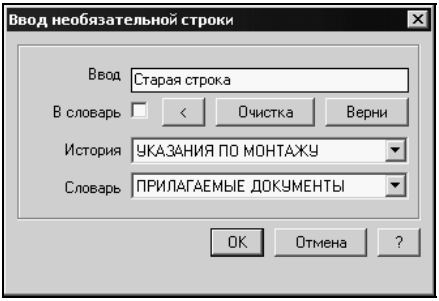


Рис. 10.4. Диалоговое окно функции `ru-dlg-dcl-get-string`. Допускается ввод пустой строки

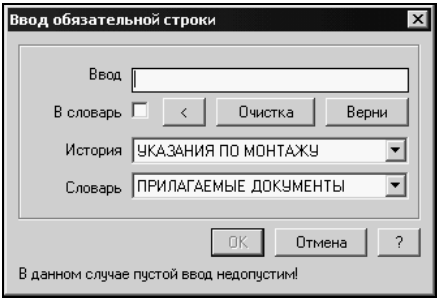


Рис. 10.5. Диалоговое окно функции `ru-dlg-dcl-get-string`. Не допускается ввод пустой строки

Листинг 10.39. Файл ru_get_string.dcl

```
// Установка уровня контроля за корректностью диалога
dcl_settings : default_dcl_settings { audit_level = 0; }
// Включаем файл, в котором определены наши стандартные элементы
@include "ru_base.dcl"
// Описываем группу inp_string
inp_string : boxed_column {
:row {
  :column {alignment = right; fixed_width = true; width = 7;
     : text {label = "Ввод"; alignment = right;}
     : text {label = "В словарь"; alignment = right;}
     : text {label = "История"; alignment = right;}
    : text {label = "Словарь"; alignment = right;}
  }
  :column {
// У поля редактирования устанавливаем атрибут allow_accept = true;
// что обеспечивает активизацию этого поля при нажатии <Enter>
     : edit_box {key = "string"; edit_limit = 256; allow_accept = true;}
     :row {alignment = right;
// У переключателя метку задаем пустой, потому что вместо ее стандартного
// расположения справа мы ввели обычный текст "В словарь" слева
          : toggle {label = ""; key = "add_file";}
// Группа from_scr определена в ru_base.dcl
          from_scr;
          : button {label = "Очистка"; key = "btn_clear";}
          : button {label = "Верни"; key = "btn_redo";}
     }
// Два раскрывающихся списка. К сожалению, в DCL
// отсутствуют комбинированные списки из поля редактирования
// и выпадающего списка
     : popup_list {key = "from_history";}
     : popup_list {key = "from_file";}
  }
 }
```

```
spacer_1;
}
// Описание диалогового окна в целом
dlg_getstr : dialog {key ="title"; label = "Ввод текстовой строки";
 initial_focus = "string";
// Мы ввели key ="title" чтобы по этому ключу программно изменять
// заголовок диалогового окна
     inp_string;
     :row { alignment = right; fixed_width = true;width = 20;
    _ok_cancel_help; // Описано в ru_base.dcl
   }
errtile; // Это стандартный групповой элемент
}
```

Пояснения к описанию диалогового окна приведены в комментариях. Теперь рассмотрим исходный текст функции (листинг 10.40). Чтобы не отвлекаться на детали реализации, просмотрите сначала те части заголовка функции и ту основную часть тела функции, которые выделены полужирным шрифтом. Потом можно разобраться и с локальными функциями.

Листинг 10.40. Функция ru-dlg-dcl-get-string

```
(defun ru-dlg-dcl-get-string (caption default disable_empty /
;;; локальные (рабочие) переменные
i index_dlg last_str last_user_dic_pos new_str result save_flag
 user_dic_file user_list what_next
;;; Имена локальных функций, объявленных как локальные переменные
 _get-string-what-next _get-string-action-tile _get-string-set-tile
_get-string-save-params _get-string-init-params
 _get-string-test-empty-str _get-string-add-to-dic
 _get-string-upd-from-file _get-string-upd-from-history
_get-string-help _get-string-clear-str _get-string-redo-str
_get-string-update-str _get-string-upd-save-flag
_get-string-clear-error-tile)
;|Примеры:
(ru-dlg-dcl-get-string "Ввод необязательной строки" "Старая строка" nil)
(ru-dlg-dcl-get-string "Ввод обязательной строки" "Старая строка" T)
|;
;;;########## Локальная функция ##########
  (defun _get-string-clear-error-tile ()
;;; Очищает строку сообщения об ошибке
    (set_tile "error" "")
 )
;;;########## Локальная функция ##########
  (defun _get-string-upd-save-flag ()
;;; Обновление строки ввода
    (if (= i "1")
      (setq save_flag T)
      (setq save_flag nil)
   )
```

```
    (mode_tile "string" 2)
 )
;;;########## Локальная функция ##########
  (defun _get-string-update-str ()
;;; Обновление строки ввода
    (_get-string-clear-error-tile)
    (setq last_str (get_tile "string"))
    (mode_tile "string" 2)
 )
;;;########## Локальная функция ##########
  (defun _get-string-redo-str ()
;;; Возвращает в строку ввода прежнее значение
    (_get-string-clear-error-tile)
    (setq last_str default)
    (set_tile "string" last_str)
    (mode_tile "string" 2)
 )
;;;########## Локальная функция ##########
  (defun _get-string-clear-str ()
;;; Очистка строки ввода
    (_get-string-clear-error-tile)
    (setq last_str "")
    (set_tile "string" last_str)
    (mode_tile "string" 2)
 )
;;;########## Локальная функция ##########
  (defun _get-string-help ()
;;; Вывод справки по работе с диалоговым окном в специальном
;;; окне (не DCL!), создание которого мы еще разберем
    (ru-dlg-view-txt-file (ru-file-help "txt\\ru-edit-str.txt"))
 )
;;;########## Локальная функция ##########
  (defun _get-string-upd-from-history (/ str)
;;; Обновление строки ввода после выбора из списка истории ввода
    (_get-string-clear-error-tile)
    (setq Str (get_tile "from_history")
          last_str (nth (atoi Str) *ru_history_list*))
    (set_tile "string" last_str)
    (mode_tile "string" 2)
 )
;;;########## Локальная функция ##########
  (defun _get-string-upd-from-file (/ str)
;;; Обновление строки ввода после выбора из словаря
    (if  user_list
      (progn
       (_get-string-clear-error-tile)
       (setq str (get_tile "from_file")
             last_user_dic_pos (atoi str)
             last_str (strcat (get_tile "string")
             " " (nth last_user_dic_pos user_list)))
      (set_tile "string" last_str)
     )
```

```
     (set_tile "error" "Нет списка слов!")
   )
    (mode_tile "string" 2)
 )
;;;########## Локальная функция ##########
  (defun _get-string-add-to-dic (/ fn f str)
;;; добавляет строку в пользовательский словарь
    (if save_flag
      (progn
       (setq str (get_tile "string"))
       (if (ru-string-add-to-user-dict "userstr.txt" str)
          (set_tile "error" "")
          (set_tile "error" "Ошибка записи в словарь")
       )
      )
   )
 )
;;;########## Локальная функция ##########
  (defun _get-string-test-empty-str (/ s)
;;; Проверка строки ввода на пустоту
    (setq s (get_tile "string"))
    (if (/= s "") (setq s (vl-string-trim " \t\n" s)))
    (if  disable_empty
      (if (= s "")
       (progn
         (mode_tile "accept" 1)
         (set_tile "error" "В данном случае пустой ввод недопустим!")
          nil
       )
       (progn
         (_get-string-clear-error-tile)
         (mode_tile "accept" 0)
         T
       )
     )
      T
   )
 )
;;;########## Локальная функция ##########
  (defun _get-string-init-params ()
;;; Инициализация переменных
  (setq last_str default)
;; Инициализация или пополнение списка истории ввода
  (if (null *ru_history_list*) (ru-string-add-to-history last_str))
    (if  (null user_list)
      (progn
       ;; Высчитываем имя файла пользовательского словаря
       (setq user_dic_file (ru-file-current-user "userstr.txt"))
       (if (not (findfile user_dic_file)); и если словаря нет
      ;; создаем его из шаблона
        (vl-file-copy (ru-file-template "userstr.txt") user_dic_file)
      )
```

```
      ;; Заполняем список строками из словаря
       (setq user_list (ru-list-read-from-file user_dic_file))
     )
   )
   ;; Вспоминаем, какая строка словаря в прошлый раз использовалась
    (setq last_user_dic_pos
      (atoi (ru-user-read-last-param "last_user_dic_pos" "0"))
   )
 )
;;;########## Локальная функция ##########
  (defun _get-string-save-params ()
    (ru-user-write-last-param
      "last_user_dic_pos"
      (itoa last_user_dic_pos)
   );_ end of ru-user-write-last-param
 );_ end of defun
;;;########## Локальная функция ##########
  (defun _get-string-set-tile ()
;;; Заполнение  полей диалога данными
    (set_tile "title" caption)
    (set_tile "string" last_str)
   ;; создаем список истории
    (start_list "from_history")
    (mapcar 'add_list *ru_history_list*)
    (end_list)
    (set_tile "from_history" "0")
   ;; Обновление словаря пользователя
    (if  user_list
      (progn
        (start_list "from_file")
        (mapcar 'add_list user_list)
        (end_list)
        (set_tile "from_file" (itoa last_user_dic_pos))
      )
   )
 )
;;;########## Локальная функция ##########
  (defun _get-string-action-tile ()
;;; Создание обработчиков событий
    (action_tile "string" "(_get-string-test-empty-str)")
    (action_tile "from_history"
      "(_get-string-upd-from-history)(_get-string-test-empty-str)"
   )
    (action_tile "from_file"
      "(_get-string-upd-from-file)(_get-string-test-empty-str)"
   )
    (action_tile "from_screen"
      "(_get-string-update-str)(done_dialog 2)"
   )
```

```
    (action_tile "add_file"
       (strcat "(setq i $value)"
               "(_get-string-upd-save-flag)"
               "(_get-string-test-empty-str)"
      )
   )
    (action_tile "accept"
      (strcat   "(_get-string-update-str)"
              "(ru-string-add-to-history last_str)"
              "(_get-string-add-to-dic)"
              "(if (_get-string-test-empty-str)"
              "(done_dialog 1))"
      )
   )
    (action_tile "cancel" "(done_dialog 0)")
    (action_tile "help" "(_get-string-help)")
    (action_tile "btn_clear"
      "(_get-string-clear-str) (_get-string-test-empty-str)"
   )
    (action_tile "btn_redo"
      "(_get-string-redo-str)(_get-string-test-empty-str)"
   )
 )
;;;########## Локальная функция ##########
  (defun _get-string-what-next (/ res)
    (mode_tile "string" 2)
    (cond
      ((= 0 what_next)
       (setq res nil)
     )
      ((= 1 what_next)
       (setq res last_str)
     )
      ((= 2 what_next)
       (setq new_str (ru-get-txt-from-dwg "Выбери надпись из рисунка"))
      ;; добавляем выбранную строку
       (if (/= last_str "")
         (setq last_str (strcat last_str " " new_str))
         (setq last_str new_str)
      )
    )
   )
    res
 );_ end of defun
 ;; ############# Главная функция ####################
;;;Производим инициализацию параметров
  (_get-string-init-params)
;; Устанавливаем заведомо невозможную причину скрытия диалога
  (setq what_next 1000)
```

```
;; Пытаемся загрузить файл диалога из отведенного для DCL каталога
  (if (> (setq index_dlg (load_dialog (ru-file-dcl "ru_get_string"))) 0)
;; Если диалог загружен без ошибок, начинаем работу
    (progn
     ;; В цикле, как минимум первый раз пытаемся показать диалог
      (while (> what_next 1)
      ;; Пытаемся показать диалог "dlg_getstr",
      ;; находящийся в загруженном DCL с номером index_dlg
        (if (new_dialog "dlg_getstr" index_dlg)
         (progn; если диалог отображен
          ;; Устанавливаем значения полей
           (_get-string-set-tile)
          ;; Проверяем, не надо ли блокировать кнопку OK
           (_get-string-test-empty-str)
          ;; Настраиваем обработчик событий элементов диалога
           (_get-string-action-tile)
          ;; Запуск ожидания событий у элементов диалога
           (setq what_next (start_dialog))
          ;; Анализ и обработка событий
           (setq result (_get-string-what-next))
        )
         (progn; Если не удалось показать диалог
           (setq what_next 0); Провоцируем выход из цикла
           (ru-msg-alert "Не могу показать диалоговое окно!")
         )
       )
     )
     ;; После окончания работы выгружаем диалог
      (unload_dialog index_dlg)
     ;; и сохраняем параметры для следующего раза
      (_get-string-save-params)
   )
    (ru-msg-alert
      (strcat "Не могу загрузить " (ru-file-dcl "ru_get_string"))
   )
 )
;; Результатом будет текст из строки ввода при выходе по OK
;; или NIL при выходе по ESC
  result
)
```

Внутри этой функции вызывается несколько пока незнакомых нам библиотечных функций с префиксом `ru-`, которые мы разберем в последующих главах. Обратите внимание, что некоторые переменные, локальные внутри основной функции, являются *псевдоглобальными* (т. е. глобальными для локальных функций). Это позволило избежать не нужного в данном случае усложнения функции при передаче параметров.

Управляющие конструкции

Управляющие конструкции в LISP также являются функциями, и забывать об этом не надо. Функция возвращает значение последнего вычисленного выражения, следовательно, и управляющие конструкции в LISP можно использовать там, где требуется получить результат выражения.

Функция *cond*

Основным средством разветвления является функция `cond`:

```
(cond (test result ...) ...)
```

Этой функции передаются несколько списков, в каждом из которых первый элемент — тестовое выражение, а последующие элементы формируют результат теста. Функция оценивает элементы списка `test`, пока не встретит элемент, возвращающий значение, отличное от `nil`. Удобно в качестве последнего тестового выражения использовать символ `T`. В этом случае результат последнего тестового выражения будет значением функции `cond` по умолчанию. Если нет ни одного истинного тестового выражения, функция вернет `nil`. В частном случае `cond` может проверять одно или два условия, т. е. заменять функцию

```
(if тест-выражение выражение-тогда [выражение-иначе])
```

Хотя функция `if` имеет более естественную форму, но `выражение-тогда` и `выражение-иначе` должны быть именно одним выражением и, при необходимости вычисления нескольких, выражения должны обрамляться функцией `progn`. А это уже лишние скобки и вызовы функции.

Простое выражение

```
(if (= (ru-acad-ver) 15)
  (setq dos_lib_arx_file "doslib2k.arx")
  (setq dos_lib_arx_file "doslib2004.arx")
);_ end of if
```

будет усложнено, если понадобится вставить сообщение:

```
(if (= (ru-acad-ver) 15)
  (progn
    (princ "\nЗагружаю doslib2k")
    (setq dos_lib_arx_file "doslib2k.arx")
  )
  (progn
    (princ "\nЗагружаю doslib2004")
    (setq dos_lib_arx_file "doslib2004.arx")
 )
)
```

А если появится версия AutoCAD R17? Лучше уж мы применим "на вырост" функцию `cond`:

```
(cond
  ((= (ru-acad-ver) 15)
   (princ "\nЗагружаю doslib2k")
```

```
   (setq dos_lib_arx_file "doslib2k.arx")
 )
  (T
   (princ "\nЗагружаю doslib2004")
   (setq dos_lib_arx_file "doslib2004.arx")
 )
)
```

Имеется еще одна интересная особенность функции `cond`, которой редко пользуются — если в проверяемом условии имеется только тестовое выражение, а результат отсутствует, то возвращается результат тестового выражения. Функция

```
(defun ru-obj-get-acad-object ()
  (cond (*ru_acad-object*)
        (t
         (setq *ru_acad-object* (vlax-get-acad-object))
        )
 )
)
```

гораздо изящнее, чем

```
(defun ru-obj-get-acad-object ()
  (if (not *ru_acad-object*)
    (setq *ru_acad-object* (vlax-get-acad-object))
 )
  *ru_acad-object*
)
```

При использовании функции `cond` следует помнить, что выход из функции происходит сразу же, как только встретится первое истинное условие, а остальные условия проверяться не будут.

О лишних функциях *progn*

Функция `progn` введена для того, чтобы объединить группу последовательных выражений в одно, там где требуется именно одно выражение, например, в функции `if`. Часто можно встретить ненужные вызовы функции `progn`. Вреда они не приносят, но являются признаком неаккуратности программиста. Иногда это последствия удаления имевшихся ранее выражений.

В системе ruCAD имеется одно место, где вызовы `progn` кажутся не нужными, но они совершенно необходимы — это "макросы" в XML-меню.

```
<item name='Влево' image='draw\u_l.gif'
comment='Уклоноуказатель с направлением влево'
macro='(progn(ru-app-load "ru_block_insert_lib")
(start "00000000" "УКЛОН_ВЛЕВО" 1 2.5 2.5 NIL))'/>
```

В этом пункте меню атрибут `macro` является строкой, в которой находится LISP-выражение, "обернутое" в функцию `progn`. Сделано это потому, что строка должна быть прочитана функцией `read`, а выражение "выполнено" функцией `eval`, которой передается одно выражение.

Циклическая обработка списков

В языке LISP, в отличие от других языков программирования, нет цикла с переменной, меняющейся от минимального до максимального значения с заданным шагом. И не очень-то надо, т. к. большей частью обрабатываются списки.

В других языках, обычно, цикл с переменной применяется для прохода по всем элементам массива и их обработки. В LISP для обработки каждого элемента списка предусмотрены две функции — `foreach` и `mapcar`.

Функция `foreach` имеет синтаксис:

```
(foreach элемент_списка список выражение)
```

Проходя по всем элементам списка, она выполняет с каждым элементом заданное выражение. Пример записи списка строк в файл приведен в листинге 10.41.

Листинг 10.41. Функция `ru-list-write-to-file`

```
(defun ru-list-write-to-file (filename string_list / f result)
  (if (setq f (open filename "w"))
    (progn
      (foreach x string_list
       (princ (strcat x "\n") f)
     )
      (setq f (close f)
           result filename)
   )
    (princ (strcat "\nНе могу создать файл \n" filename))
 )
  result
)
```

Для прохода по всем членам семейства объектов имеется аналогичная функция `vlax-for` (листинг 10.42).

Обратите внимание — переменная `x` (элемент списка) не объявляется локальной. В этом нет необходимости: переменные, используемые функциями `foreach` и `vlax-for`, фактически эквивалентны аргументам, задаваемым при объявлении функций через функции `defun` или `lambda`. Разница лишь в упрощенном синтаксисе при объявлении, поскольку в этих конструкциях всегда только один аргумент и не может быть локальных переменных.

Листинг 10.42. Функция `ru-obj-get-styles-names-fonts`

```
(defun ru-obj-get-styles-names-fonts (/ result)
  (vlax-for Sty (vla-get-textstyles (ru-get-active-document))
    (setq result (cons
                   (cons (vla-get-name Sty) (vla-get-fontfile Sty))
                     result
                   )
   )
 )
  (reverse result)
)
```

Для выполнеия каких-то действий над всеми элементами списка применяется функция mapcar. Простейший пример формирования списка строк для раскрывающегося списка диалогового окна

```
(start_list "from_file")
(mapcar 'add_list user_list)
(end_list)
```

В этом примере mapcar выполняет функцию add_list поочередно со всеми элементами списка user_list. В данном случае элементы списка просто добавляются в поле с именем "from_file". Обратите внимание, что перед add_list стоит апостроф, т. к. функции mapcar должно быть передано символьное имя применяемой функции, а не результат ее выполнения.

Функция возвращает список результатов выполнения примененной функции.

Рассмотрим более сложный пример.

Листинг 10.43. Функция ru-list-massoc

```
(defun ru-list-massoc (key alist)

;;; Пример:(ru-list-massoc 10 (entget (car (entsel))))
(mapcar 'cdr (vl-remove-if-not (function (lambda (x) (= key (car x)))) alist))
);_ end of defun
Command: (ru-list-massoc 10 (entget (car (entsel))))
Select object: ((119.726 165.816) (175.252 377.274) (337.2 237.457) (452.878
252.479) (519.971 393.451) (622.924 441.983) (839.241 253.634) (715.466
129.995) (435.526 164.66) (295.556 121.906) (253.912 173.904))
```

Эта функция является примером LISP-технологии. В ней применены самые "хитрые" функции — и mapcar, и function, и lambda. Функции передается целочисленный код и ассоциированный список, а возвращается список из данных элементов исходного списка, у которых ключ соответствует заданному. В примере использования в функцию ru-list-massoc был передан ключ 10 и ассоциированный список данных примитива, выглядевший так:

```
((-1 . <Entity name: 40074e70>) (0 . "LWPOLYLINE")
(330 . <Entity name: 40074cf8>) (5 . "76") (100 . "AcDbEntity")
(67 . 0) (410 . "Model") (8 . "0") (100 . "AcDbPolyline") (90 . 11)
(70 . 128) (43 . 0.0) (38 . 0.0) (39 . 0.0) (10 119.726 165.816)
(40 . 0.0) (41 . 0.0) (42 . 0.0) (10 175.252 377.274)
(40 . 0.0) (41 . 0.0) (42 . 0.0) (10 337.2 237.457) (40 . 0.0)
(41 . 0.0) (42 . 0.0) (10 452.878 252.479) (40 . 0.0) (41 . 0.0)
(42 . 0.0) (10 519.971 393.451) (40 . 0.0) (41 . 0.0) (42 . 0.0)
(10 622.924 441.983) (40 . 0.0) (41 . 0.0) (42 . 0.0)
(10 839.241 253.634) (40 . 0.0) (41 . 0.0) (42 . 0.0)
(10 715.466 129.995) (40 . 0.0) (41 . 0.0) (42 . 0.0) (10 435.526 164.66)
(40 . 0.0) (41 . 0.0) (42 . 0.0) (10 295.556 121.906) (40 . 0.0)
(41 . 0.0) (42 . 0.0) (10 253.912 173.904) (40 . 0.0) (41 . 0.0)
(42 . 0.0) (210 0.0 0.0 1.0))
```

Из этого списка были извлечены подсписки, у которых DXF-код равен 10, т. е. мы получили список вершин полилинии. Но функцию можно применять не только для списков данных примитивов.

```
Command:
(ru-list-massoc 10 (list '(10 . "Вентилятор") '(20 . "В-Ц-14-46")
'(10 . "Вентиль") '(20 . "15Б1бк")))
("Вентилятор" "Вентиль")
Command:
(ru-list-massoc 20 (list '(10 . "Вентилятор") '(20 . "В-Ц-14-46")
'(10 . "Вентиль") '(20 . "15Б1бк")))
("В-Ц-14-46" "15Б1бк")
(ru-list-massoc 30 (list '(10 . "Вентилятор") '(20 . "В-Ц-14-46")
'(10 . "Вентиль") '(20 . "15Б1бк")))
nil
```

Разберемся, как же работает эта функция. Функция `vl-remove-if-not` применяет функцию `lambda` (не вычисляя ее результат) ко всем элементам списка и удаляет все ненужные элементы. Функция `mapcar` применяет функцию `cdr` (не вычисляя ее результат) к результату, возвращенному функцией `vl-remove-if-not`, а это список, как мы уже знаем.

Функция `lambda` создает "одноразовую" функцию, которая в данном случае имеет вид

```
(lambda (x) (if (= key (car x)) x))
```

Можно было бы определить функцию через `defun`, но это было бы сложно и не элегантно. Внутри одноразовой функции видны текущие локальные переменные, в данном случае — `key`. Извлекать его текущее значение для текущего элемента списка было бы затруднительно. Как и при определении функции при помощи `defun`, "одноразовая" функция возвращает результат вычисления последнего выражения, т. е. в нашем примере "хвост" подсписка, если голова подсписка совпадает со значением `key`.

Подобные приемы работы часто будут применяться в наших функциях.

Рекурсия

Одним из самых интересных механизмов, реализованных в языке LISP, является *рекурсия*[1]. Рекурсивные вызовы функций считаются одним из основных приемов этого языка. Однако блестящее решение на языковом уровне может быть сведено "на нет" конкретными реализациями. Любые рекурсивные конструкции преобразуются транслятором в команды конкретного процессора, а он оперирует другими понятиями. О применении рекурсии в конкретных системах до сих пор идут споры. Программисты со стажем помнят категоричные рекомендации для AutoLISP времен версии 10: "Избегайте рекурсий!" Главной проблемой тех времен был мизерный объем доступной памяти и размер стека. С тех пор ситуация изменилась. Проблем с памятью практически нет, но суеверия по отношению к рекурсии остались.

---

[1] Рекурсия — это такая организация алгоритма, при которой процедура обращается к самой себе. — *Ред.*

Расследование Петра Лоскутова

Итак, *рекурсия*, заключается она в возможности функции вызывать саму себя. Выглядит это примерно так:

Листинг 10.44. Структура рекурсивной функции

```
(defun my-test-function (arg)
 (if <условие>
   (my-test-function (<некая функция> arg))
   <действие при невыполненном условии>
 );_ end of if
);_ end of defun
```

На самом деле, возможность рекурсии логично вытекает из идеологии LISP — если возможен вызов функции внутри другой функции, то почему это не может быть та же самая функция. Тем не менее, используется рекурсия LISP-программистами не слишком широко и далеко не всегда оправдано. Причин тому несколько:

- чаще всего рекурсия используется для обработки списков, а LISP имеет очень развитые встроенные функции, прямо предназначенные для этого;
- читабельность исходного текста рекурсивной функции зачастую существенно хуже, чем при применении итерации или встроенной функции;
- некоторые еще помнят давнишнюю рекомендацию Autodesk о нежелательности применения рекурсии[1];
- действительно имеющиеся ограничения, в том числе технические, при применении данного способа;
- недостаточная компетентность многих LISP-программистов, особенно занимающихся программированием помимо основной специальности;
- некоторая мифологизированность этой темы.

Рассматривать механизм рекурсии в LISP невозможно без сравнения с итерационными алгоритмами и встроенными функциями и делать это надо на примерах. Для начала возьмем простейший вариант обработки списка, чтобы показать различия в реализации с помощью разнообразных методов.

Попробуем сравнить различные подходы и определить области целесообразности их применения. Пусть у нас есть предельно простой линейный список вида: (1 2 3 4 5), из каждого члена которого мы хотим извлечь квадратный корень и результат получить в виде списка. Реализация разными способами может быть такой:

Листинг 10.45. Сравнение реализаций простой обработки простейшего списка

```
(defun lst-sqrt-map (lst)
   ;; Используется встроенная функция 'mapcar'.
    (mapcar 'sqrt lst)
);_ end of defun
```

[1] Есть подозрения, что связано это было не только и не столько с техническими ограничениями, сколько с усталостью службы техподдержки Autodesk от объяснений причин сбоев программ, написанных непрофессиональными LISP-программистами.

```
(defun lst-sqrt-iter (lst / item tmp)
   ;; Итерационная функция.
    (while (setq item (car lst))
        (setq tmp (cons (sqrt item) tmp)
              lst (cdr lst)
       );_ end of setq
   );_ end of while
    (reverse tmp)
);_ end of defun

(defun lst-sqrt-rec (lst)
   ;; Рекурсивная функция.
    (if lst
        (cons (sqrt (car lst)) (lst-sqrt-rec (cdr lst)))
   );_ end of if
);_ end of defun

(setq test-list '(1 2 3 4 5))
;;; Присвоим переменной 'test-list' наш список.
;;; Проверим выполнение:
;;; Встроенная функция:
(lst-sqrt-map test-list)
(1.0 1.41421 1.73205 2.0 2.23607)
;;; Итерационная функция:
(lst-sqrt-iter test-list)
(1.0 1.41421 1.73205 2.0 2.23607)
;;; Рекурсивная функция:
(lst-sqrt-rec test-list)
(1.0 1.41421 1.73205 2.0 2.23607)
;;; Получаем одинаковый результат.
```

Очевидно, что в этом случае применение встроенной функции `mapcar` несоизмеримо проще других вариантов. Это не удивительно, т. к. именно для подобных случаев данная функция и предназначена в первую очередь. Отметим только, что итерационный код оказался наибольшего объема и потребовал введения локальных переменных, отсутствующих в других решениях. Это вполне характерная ситуация: грамотное использование встроенных функций и/или рекурсии, как правило, позволяет уменьшить число локальных переменных по сравнению с итерационными алгоритмами.

Посмотрим, что произойдет, если обработка элементов списка должна производиться "по условию" (листинг 10.46). Предположим, что в списке, подобном предыдущему, мы хотим все элементы, кратные "3", разделить на "2", четные, кроме кратных "3", разделить на "3", а из остальных извлечь квадратный корень.

Листинг 10.46. Обработка простейшего списка "по условию"

```
(defun lst-sqrt-map-2 (lst)
   ;; Используется встроенная функция 'mapcar'
    (mapcar
```

```
        (function
            (lambda (item)
                (cond
                    ((= 0 (rem item 3)) (/ item 2.0))
                    ((= 0 (rem item 2)) (/ item 3.0))
                    (T (sqrt item))
                );_ end of cond
            );_ end of lambda
        );_ end of function
        lst
    );_ end of mapcar
);_ end of defun

(defun lst-sqrt-iter-2 (lst / item tmp)
    ;; Итерационная функция
    (while (setq item (car lst))
        (setq tmp (cons (cond
                            ((= 0 (rem item 3)) (/ item 2.0))
                            ((= 0 (rem item 2)) (/ item 3.0))
                            (T (sqrt item))
                        );_ end of cond
                        tmp
                    );_ end of cons
              lst (cdr lst)
        );_ end of setq
    );_ end of while
    (reverse tmp)
);_ end of defun

(defun lst-sqrt-rec-2 (lst / item)
    ;; Рекурсивная функция
    (if lst
        (cons (cond
                  ((= 0 (rem (setq item (car lst)) 3)) (/ item 2.0))
                  ((= 0 (rem item 2)) (/ item 3.0))
                  (T (sqrt item))
              );_ end of cond
              (lst-sqrt-rec-2 (cdr lst))
        );_ end of cons
    );_ end of if
);_ end of defun

(setq test-list-2 '(1 2 3 4 5 6 7 8 9 10))

;;; Проверим выполнение:
(lst-sqrt-map-2 test-list-2)
(1.0 0.666667 1.5 1.33333 2.23607 3.0 2.64575 2.66667 4.5 3.33333)
;;; Остальные функции вернут то же самое.
```

Здесь ситуация уже неочевидная. Сложность реализации практически идентична, результат — тоже. Не видно критерия, на основании которого можно сделать выбор. Попробуем поискать его в сравнении производительности.

Примечание

При написании LISP-программ скорость выполнения собственно кода редко бывает критичной, особенно на современной технике. Как правило, значительно больше времени отнимают ввод данных пользователем, выбор им примитивов, которые должны участвовать в работе программы, перерисовка экрана и отображение результатов. Тем не менее, оптимизация по скорости выполнения программы совсем не лишнее дело, тем более что при развитии любой системы, обычно, происходит ее разрастание и функция, которая сегодня используется однократно, завтра может оказаться частью другой функции и может применяться, например, в многократном цикле.

Для выяснения быстродействия разных реализаций устроим функциям, приведенным в листинге 10.46, "гонки". Для этого нам понадобится функция, многократно вызывающая тестируемую функцию с фиксацией времени выполнения и функция для генерации длинных списков, которые будут использоваться в тестировании (листинг 10.47).

Листинг 10.47. Инструменты тестирования рекурсивных функций

```
(defun bench-lst (tst-func tst-arg iter / t-start t-stop t-tmp
                                          t-bench minut zin)
 (tst-func tst-arg);_ Предварительный прогон
 (setq t-start (getvar "DATE")
       zin     (getvar "DIMZIN")
);_ end of setq
 (setvar "DIMZIN" 1)
 (repeat iter (tst-func tst-arg))
  (setq t-stop  (getvar "DATE")
        t-tmp   (* 86400.0 (- t-stop t-start))
        t-bench
           (if (< 60 t-tmp)
               (strcat (itoa (setq minut (fix (/ t-tmp 60))))
                       ":"
                       (if (= 1
                              (vl-string-position 46
                                 (setq t-tmp
                                       (rtos (- t-tmp (* 60 minut)) 2 2)
                                          );_ end of setq
                                 );_ end of vl-string-position
                              );_ end of =
                              (strcat "0" t-tmp)
                                      t-tmp
                          );_ end of if
                    );_ end of strcat
                    (strcat
                        "0:"
                        (if (= 1
                               (vl-string-position 46
                                  (setq t-tmp (rtos t-tmp 2 2))
                               );_ end of vl-string-position
                            );_ end of =
                            (strcat "0" t-tmp)
```

```
                        t-tmp
                    );_ end of if
                );_ end of strcat
            );_ end of if
 );_ end of setq
 (setvar "DIMZIN" zin)
 (princ "\n**************Отчет**************")
 (princ "\n---------------------------------")
 (princ (strcat "\nДлина списка данных:     " (itoa (length tst-arg))))
 (princ (strcat "\nИтераций теста (раз):     " (itoa iter)))
 (princ (strcat "\nРезультат (мин:сек.мсек): " t-bench))
 (princ "\n---------------------------------\n")
 (princ)
);_ end of defun

(defun list-repeat (lst iter / tmp)
    (repeat iter (setq tmp (append tmp lst)))
);_ end of defun
```

Этапы тестирования описаны в листинге 10.48.

Листинг 10.48. Тестирование некомпилированных функций

```
;;; Используем в качестве исходного список из предыдущего примера.
;;; Используем некомпилированные версии функций.
;;; Загрузим все необходимые функции и приступим к тестированию.
;;; Встроенная функция:
(bench-lst lst-sqrt-map-2 (list-repeat test-list-2 200) 5000)
Результат (мин:сек.мсек): 0:56.93
;;; Итерационная функция:
(bench-lst lst-sqrt-iter-2 (list-repeat test-list-2 200) 5000)
Результат (мин:сек.мсек): 1:14.77
;;; Рекурсивная функция:
(bench-lst lst-sqrt-rec-2 (list-repeat test-list-2 200) 5000)
Результат (мин:сек.мсек): 1:21.90
* Примечание: оставлены только интересующие данные.
```

Общие пояснения к организации тестирования

Проводить тестирование надо на компьютере, на котором устойчиво работает система AutoCAD. Во время тестирования на компьютере не должны выполняться никакие задачи в фоновом режиме (мультимедийные программы, архиваторы, клиенты распределенных вычислений и тому подобное). Пользовательские программы, кроме AutoCAD, должны быть закрыты. Система AutoCAD должна быть полностью загружена, а компьютер не должен выполнять никаких операций с диском. Загрузку и выполнение тестов следует производить непосредственно в AutoCAD, а не в редакторе Visual LISP. Запуск тестов должен осуществляться с клавиатуры в командной строке, а мышь не должна перемещаться во время всего выполнения каждого теста. Для оценки быстродействия выполнения теста не следует использовать первый прогон каждого теста.

Не надо забывать, что время выполнения тестовой программы и время, измеренное как тестовое, совершенно разные вещи. Работая с большими списками, AutoCAD гораздо

больше времени тратит на "приборку" при завершении программы, чем непосредственно на ее выполнение[1].

Посмотрим, что же мы получили в результате тестирования (листинг 10.48). По быстродействию встроенная функция заметно опередила другие варианты. Если принять за 100% время выполнения теста встроенной функцией, то итерационная затратила на 31% больше времени, а рекурсивная — на 44%. Но и это еще не все, мало того, что интересующая нас рекурсивная функция показала самый плохой результат по времени выполнения, она еще имеет ограничения по размеру обрабатываемого списка. Эти ограничения являются аппаратно-зависимыми и для компьютера, на котором производилось тестирование, такой предел оказался практически точно равен 20 тысячам элементов.

Если все так печально, то зачем вообще нужен механизм рекурсии? Попробуем повторить все то же самое, но загрузив откомпилированные функции (листинг 10.49).

Листинг 10.49. Тестирование компилированных функций

```
;;; Условия полностью идентичны предыдущему тесту.
;;; Встроенная функция:
(bench-lst lst-sqrt-map-2 (list-repeat test-list-2 200) 5000)
Результат (мин:сек.мсек): 0:27.87
;;; Итерационная функция:
(bench-lst lst-sqrt-iter-2 (list-repeat test-list-2 200) 5000)
Результат (мин:сек.мсек): 0:27.93
;;; Рекурсивная функция:
(bench-lst lst-sqrt-rec-2 (list-repeat test-list-2 200) 5000)
Результат (мин:сек.мсек): 0:30.15
```

Картина претерпела изрядные изменения. Распределение мест по быстродействию не изменилось, но разрыв между лидером и аутсайдером составил не свыше 40%, а всего около 8%, что уже не представляется столь критичным. Помимо этого, ограничение на размер обрабатываемого списка для рекурсивной функции сдвинулось куда-то за 100 тысяч элементов. На однородных списках, содержащих большее количество элементов, тестирование не проводилось, ввиду их экстремальной экзотичности.

Примечание

Хочется искренне поблагодарить группу программистов фирмы Autodesk, занимавшуюся созданием LISP-компилятора. Столь серьезный выигрыш по скорости выполнения компилированных функций по сравнению с некомпилированными и столь малая разница в скорости выполнения встроенных и пользовательских функций говорят об очень грамотной оптимизации компиляции. Также стоит отметить, что переход системы AutoCAD на исполнение пользовательских функций, "компилируемых" из исходных текстов непосред-

[1] Ввиду того, что все тесты выполнялись в аппаратной конфигурации, гарантированно позволяющей провести тестирование от начала до конца без сбора "мусора" в процессе выполнения, данному вопросу внимание не уделялось. При невозможности обеспечить подобные условия необходимо позаботиться о ручном сборе "мусора" с вычетом времени, затраченного на "приборку".

ственно в момент загрузки, не надо понимать, как альтернативу полноценной компиляции[1].

Учитывая очевидное положительное влияние компиляции на производительность функций и не найдя никаких причин, препятствующих ее использованию, все дальнейшие тесты мы будем проводить только на компилированных функциях. Параметры компиляции на производительности функций не сказываются, по крайней мере автором данного материала эффекта от компиляции в разных режимах даже достаточно крупных модулей ни разу замечено не было. Что, впрочем, легко объяснимо, т. к. предназначены они совсем для других целей.

Тем не менее пока не видно никаких аргументов *в пользу* применения рекурсии. Проигрывает она после компиляции не слишком много, но проигрывает. Попробуем изменить условия задачи — предположим, что нам нужно обработать аналогичным предыдущим тестам образом неоднородный список, т. е. список, элементами которого могут быть как атомы, так и списки, в которых, в свою очередь, так же могут быть и атомы, и списки. При этом структуру списка будем считать неизвестной. В качестве "подопытного" используем такой список:

```
(1 2 (3 4 5) (6 7 (8 9 10 (11 12 13 14 15 16))) 17 18 19 20)
```

Обрабатывать его будем по условиям предыдущего теста. Вот тут-то и становится ясно, что написать функцию для обработки такого списка, без применения рекурсии, по меньшей мере, затруднительно. Желающие могут попрактиковаться в навыках написания таких функций с применением только встроенных или итерационных функций (а несмотря на "неудобность" сложного списка, обработать его без рекурсии все же возможно), а мы посмотрим, как это будет выглядеть для рекурсивной функции (листинг 10.50).

Листинг 10.50. Рекурсивная функция для обработки неоднородных списков

```
 (defun lst-sqrt-rec-3 (lst / item)
;; Рекурсивная функция
 (if lst
     (cons (cond
               ((= 'list (type (setq item (car lst))))
                              (lst-sqrt-rec-3 item))
               ((= 0 (rem item 3)) (/ item 2.0))
               ((= 0 (rem item 2)) (/ item 3.0))
               (T (sqrt item))
          );_ end of cond
           (lst-sqrt-rec-3 (cdr lst))
    );_ end of cons
);_ end of if
);_ end of defun
(setq test-list-3 '(1 2 (3 4 5) (6 7 (8 9 10 (11 12 13 14 15 16))) 17 18 19 20))
```

[1] Чтобы не вносить дополнительную путаницу в терминологию, не будем вдаваться в рассуждения о "полноценности компиляции" в AutoLISP, с точки зрения других языков программирования, для нас важна ее эффективность.

Проверим выполнение:

```
(lst-sqrt-rec-3 test-list-3)
(1.0 0.666667 (1.5 1.33333 2.23607) (3.0 2.64575 (2.66667 4.5 3.33333 (3.31662 6.0
3.60555 4.66667 7.5 5.33333))) 4.12311 9.0 4.3589 6.66667)
```

Фактически понадобилось добавить всего одну строку и одну отредактировать, чтобы рекурсивная функция смогла обрабатывать неоднородный список с неизвестной заранее структурой. Можно совершенно определенно сказать, что никаким иным способом достичь подобной компактности невозможно. Вот мы и увидели, по крайней мере, одну ситуацию, когда применение рекурсивных функций не только оправдано, но, как правило, является наилучшим решением. Попробуем представить, в какой еще ситуации может быть полезной рекурсия.

Предположим, что требуется изменить значение элемента с известным порядковым номером в однородном списке. Встроенные функции здесь неприменимы, точнее — применимы, но в изрядно извращенном виде. Разумно задачу можно решить либо с помощью итерации, либо рекурсии (листинг 10.51).

Листинг 10.51. Замена элемента списка с известным номером

```
(defun lst-item-chg-iter (_mli _mtl _val / _tmp)
   ;; Типичная итерационная функция
    (repeat (1- _mli)
        (setq _tmp (cons (car _mtl) _tmp)
              _mtl (cdr _mtl)
        );_ end of setq
   );_ end of repeat
    (setq _tmp (cons _val _tmp)
          _mtl (cdr _mtl)
   );_ end of setq
    (while _mtl
        (setq _tmp (cons (car _mtl) _tmp)
              _mtl (cdr _mtl)
        );_ end of setq
   );_ end of while
    (reverse _tmp)
);_ end of defun

(defun lst-item-chg-rec (_mli _mtl _val)
   ;; Переименованная библиотечная
    (if (> _mli 1)
        (cons (car _mtl) (lst-item-chg-rec (1- _mli) (cdr _mtl) _val))
        (cons _val (cdr _mtl))
   );_ end of if
);_ end of defun

(setq test-list-2 '(1 2 3 4 5 6 7 8 9 10))
```

Проверим выполнение:

```
(lst-item-chg-rec 7 test-list-2 "Test")
(1 2 3 4 5 6 "Test" 8 9 10)
(lst-item-chg-iter 7 test-list-2 "Test")
(1 2 3 4 5 6 "Test" 8 9 10)
```

Рекурсивная функция оказалась значительно компактнее. Следует заметить, что итерационную функцию можно существенно уменьшить, но за счет быстродействия. Ничего не принимая "на веру", убедимся, в этом (листинг 10.52).

Листинг 10.52. Укороченная итерационная функция

```
(defun lst-item-chg-iter-2 (_mli _mtl _val / _tmp _item _iter)
 ;; Итерационная функция
 (setq _iter 0)
 (while _mtl
   (setq _item (car _mtl)
      _mtl (cdr _mtl)
      _iter (1+ _iter)
      _tmp (cons (if (= _iter _mli) _val _item) _tmp)
   );_ end of setq
 );_ end of while
 (reverse _tmp)
);_ end of defun
```

Модифицируем функцию тестирования скорости выполнения так, чтобы она отвечала изменившимся условиям (исходный текст здесь не приводится), откомпилируем все функции, загрузим их и проведем тестирование (листинг 10.53).

Листинг 10.53. Второе тестирование компилированных функций

```
;;; Итерационная функция, 10000 элементов:
(bench-lst-2 'lst-item-chg-iter (list 10 (list-repeat test-list-2 1000) "Test") 1000)
Результат (мин:сек.мсек): 0:22.06
;;; Укороченная итерационная функция, 10000 элементов:
(bench-lst-2 'lst-item-chg-iter-2 (list 10 (list-repeat test-list-2 1000) "Test") 1000)
Результат (мин:сек.мсек): 0:26.48
;;; Рекурсивная функция, 10-й элемент из 10000:
(bench-lst-2 'lst-item-chg-rec (list 10 (list-repeat test-list-2 1000) "Test") 1000)
Результат (мин:сек.мсек): 0:00.06
;;; Рекурсивная функция, 5000-й элемент из 10000:
(bench-lst-2 'lst-item-chg-rec (list 5000 (list-repeat test-list-2 1000) "Test") 1000)
Результат (мин:сек.мсек): 0:11.03
;;; Рекурсивная функция, 10000-й элемент из 10000:
(bench-lst-2 'lst-item-chg-rec (list 10000 (list-repeat test-list-2 1000) "Test") 1000)
Результат (мин:сек.мсек): 0:23.16
```

Тут мы столкнулись с еще одной особенностью рекурсивных функций. В случае обращения к элементу "по номеру" скорость выполнения рекурсивной функции становится зависимой от местоположения искомого элемента. При этом обращений к элементам списка, стоящим после искомого, не происходит. Соответственно, очень сильно меняется и время выполнения функции. В таких случаях, как правило, оперируют средним временем, т. е. при тестировании списка из 10 тысяч элементов это обращение к 5 000-му элементу.

Справедливости ради, нужно сказать, что подобный механизм несложно реализовать и для итерационной функции, и иногда, именно это будет оптимальным решением, но, во-первых, для рекурсивной этот прием естественен, исходя из самого принципа рекурсии, во-вторых, применение данного приема в итерационной функции все равно приведет к ее проигрышу, хотя и незначительному, в сравнении с рекурсивной по скорости, при параметрах запуска, позволяющих надежно фиксировать результат. Будучи людьми, от природы недоверчивыми, проведем проверку и этого утверждения (листинг 10.54).

Листинг 10.54. Еще одна итерационная функция

```
(defun lst-item-chg-iter-3 (_mli _mtl _val / _tmp)
   ;; Итерационная функция
    (repeat (1- _mli)
        (setq _tmp (cons (car _mtl) _tmp)
              _mtl (cdr _mtl)
        );_ end of setq
   );_ end of repeat
    (setq _mtl (cons _val (cdr _mtl)))
    (while _tmp
        (setq _mtl (cons (car _tmp) _mtl)
              _tmp (cdr _tmp)
        );_ end of setq
   );_ end of while
    _mtl
);_ end of defun
;;; Итерационная функция, 5000-й элемент из 10000:
Command: (bench-lst-2 'lst-item-chg-iter-3 (list 5000 (list-repeat test-list-2 1000)
"Test") 1000)
Результат (мин:сек.мсек): 0:12.27
```

Как видим, проигрыш, в сравнении с рекурсией, составил чуть более 10%. Это, конечно, не так много, чтобы сделать такой вариант неприемлемым. Все же, в сочетании с более компактным кодом, рекурсивная функция здесь выглядит предпочтительней. В целом, по результатам данного теста, можно сделать справедливый вывод о примерно двукратном превосходстве в скорости рекурсивной функции над типичной итерационной, если требуется изменить значение элемента списка по его расположению в списке. Но не стоит забывать и о приемах, позволяющих существенно снизить среднее время выполнения итерационных функций.

Что касается на 20% более медленного выполнения короткой итерационной функции в данном тестировании, чем длинного, то, не вдаваясь в подробности, объясняется это "лишней" проверкой условия внутри цикла.

На основании всех приведенных примеров и рассуждений можно сделать вывод, что при обработке нерегулярных списков, как правило, следует применять рекурсивные функции. В случае обработки регулярных списков наилучший результат дают встроенные функции, за исключением ситуаций, когда при обработке элементов списка необходимо учитывать результаты, полученные на предыдущих итерациях. Последнее утверждение можно обосновать следующей ситуацией. Если при обработке списка необходимо, при выполнении некоторого условия, принять решение о прекра-

щении обработки, то встроенная функция для этой цели не подойдет[1]. Придется воспользоваться либо итерацией, либо рекурсией.

Безотносительно же к приведенным примерам, не стоит отбрасывать ни один из продемонстрированных механизмов. Во всех случаях наилучшие результаты можно получить, разумно сочетая их все, нередко в рамках одной функции. Попробуем продемонстрировать это на том же примере обработки нерегулярного списка. Создадим функцию, использующую и встроенную функцию, и механизм рекурсии (листинг 10.55).

Листинг 10.55. Комбинированная функция

```
(defun lst-sqrt (lst)
 ;; Используется встроенная функция 'mapcar' и рекурсия
 (mapcar
   (function
     (lambda (item)
       (cond
         ((= 'LIST (type item)) (lst-sqrt item))
         ((= 0 (rem item 3)) (/ item 2.0))
         ((= 0 (rem item 2)) (/ item 3.0))
         (t (sqrt item))
       );_ end of cond
     );_ end of lambda
   );_ end of function
   lst
 );_ end of mapcar
);_ end of defun

(setq test-list-3 '(1 2 (3 4 5) (6 7 (8 9 10 (11 12 13 14 15 16))) 17 18 19 20))
```

Проверим выполнение:

```
(lst-sqrt test-list-3)
(1.0 0.666667 (1.5 1.33333 2.23607) (3.0 2.64575 (2.66667 4.5 3.33333 (3.31662 6.0
3.60555 4.66667 7.5 5.33333))) 4.12311 9.0 4.3589 6.66667)
```

Мы легко получили функцию, одновременно использующую и встроенную функцию, и механизм рекурсии. В этом случае, мы сможем за счет резкого уменьшения количества рекурсивных вызовов снизить вероятность проявления аппаратных ограничений практически до нуля, даже для некомпилированной функции. Поскольку ничего не дается "бесплатно", то в данном случае нам приходится пожертвовать производительностью при обработке сложных списков, причем, чем сложнее список, тем больше будет проигрыш комбинированной функции (листинг 10.56).

Листинг 10.56. Сравнение рекурсивной и комбинированной функций

```
(setq test-list-2 '(1 2 3 4 5 6 7 8 9 10))
(setq test-list-3 '(1 2 (3 4 5) (6 7 (8 9 10 (11 12 13 14 15 16))) 17 18 19 20))
```

[1] Варианты с применением `exit/quit` не рассматривались ввиду их абсолютно очевидной нецелесообразности.

```
;;; Рекурсивная функция, сложный список:
Command: (bench-lst lst-sqrt-rec-3 (list-repeat test-list-3 200) 5000)
Результат (мин:сек.мсек): 2:14.74
;;; Комбинированная функция, сложный список:
Command: (bench-lst lst-sqrt (list-repeat test-list-3 200) 5000)
Результат (мин:сек.мсек): 3:38.56
;;; Рекурсивная функция, простой список:
Command: (bench-lst lst-sqrt-rec-3 (list-repeat test-list-2 200) 5000)
Результат (мин:сек.мсек): 0:56.81
;;; Комбинированная функция, простой список:
Command: (bench-lst lst-sqrt (list-repeat test-list-2 200) 5000)
Результат (мин:сек.мсек): 0:56.83
```

Как мы видим, при практически равных результатах (разница не выходит за пределы погрешности измерений) скорости обработки простого списка комбинированная функция сильно (в приведенном примере, свыше 62%) проигрывает в обработке далеко не самого сложного. Из чего можно сделать вывод, что такой вариант хорошо подходит для случая, когда в обрабатываемом списке иногда могут встречаться подсписки, которые нужно обрабатывать по тем же правилам, что и основной список, особенно если основной список, хотя бы теоретически, может быть очень большим.

Вне рассмотрения остался интересный вариант рекурсии, когда две (или более) функции связаны в одну систему. На уровне схемы это выглядит так: есть две (ограничимся двумя) функции "A" и "B". Функция "A" может вызывать сама себя и функцию "B", а функция "B" так же может вызывать и себя, и соответственно, функцию "A". Для разумной демонстрации этого приема необходимо разработать и показать целую систему обработки специфичных сложно структурированных данных, что делает такую демонстрацию затруднительной в рамках книги, которая, в общем-то, немного о другом. Тем не менее, упомянуть такой вариант имеет смысл, как возможный, хотя и весьма редко целесообразный.

Выводы

Механизм рекурсии, предлагаемый языком LISP, позволяет решать задачи по обработке списков сложных типов, доступа к конкретным элементам списка, обеспечивая при этом очень компактный код. Возможности рекурсии на этом, конечно, не исчерпываются, задачи показать все возможные варианты применения и не было. Важно знать об особенностях этого метода и применять его тогда, когда это применение позволяет решить конкретную задачу проще и удобнее, чем без него. Безусловно, это не панацея, и использование рекурсии может быть неразумным, а в каких-то ситуациях может сделать программу неработоспособной, но отказываться от столь интересного механизма только потому, что при его применении могут возникнуть проблемы, не стоит.

Сомневающимся в том, что им придется обрабатывать сложные списки, можно порекомендовать взглянуть на структуру любого сложного XML-файла. Учитывая тенденции перевода все большего числа видов данных в этот формат, наверняка большая часть сомнений быстро отпадет.

Резюме авторов

Петр Лоскутов провел детальное исследование вопроса, и его рекомендациями мы будем руководствоваться при разработке нашей системы, однако стоит немного разъяснить нашу позицию в отношении языка XML.

В последующих главах мы будем очень много расхваливать XML и практически использовать этот язык. Тем не менее, мы ни разу не будем читать XML-файлы средствами LISP. В нашей конкретной системе это просто не нужно. В иных реализациях LISP (не для AutoCAD) обработка XML уже реализована и, разумеется, рекурсивными методами.

Если бы, как мы уже писали, развитие компьютерных технологий пошло иным путем и операционные системы писались бы на LISP (был такой шанс), то и надобности в XML просто бы не возникло, потому что для подобных случаев в LISP имеются ассоциированные списки. Сам язык и создавался для работы со списками! Замените в XML угловые скобки на круглые и вы получите LISP-список, только ухудшенный. Язык XML придуман для программ, не умеющих обрабатывать списки так легко, как это можно сделать в LISP.

Но, как только мы выйдем за рамки САПР (не в этой книге), разборка XML средствами AutoLISP непременно потребуется. Вот конкретная задача, над которой мы сейчас работаем — хранение пространственных данных в базе данных Interbase с возможностью отображения векторных карт в Интернете и экспортом данных в несколько разноплатформенных ГИС, в том числе базирующихся на AutoCAD. Лучшим стандартным экспортным форматом, безусловно, является XML. Вот для такой задачи потребуется разработка XML-парсера на LISP. Сделать же такой разборщик на LISP гораздо легче, чем на любом другом языке. За исключением такого маленького нюанса — для всех других языков они уже сделаны, а мы только собираемся. А это "две большие разницы".

Обработка ошибок

До сих пор мы говорили о предотвращении ошибок. Однако не все ошибки можно предотвратить путем ухищрений с исходным текстом. Многие ошибки возникают во время работы программы.

Ошибки прерывания

Пользователь в любой момент может прервать программу нажатием клавиши <Esc>. Это "лом, против которого нет приема"[1]. Компетентные пользователи, привыкшие работать с грамотно написанными программами, в этом не нуждаются. Однако клавиша <Esc> так часто используется для отказа от каких-то действий, что многие жмут ее просто инстинктивно. Как мы уже упоминали, встречаются и плохие LISP-программы, полностью построенные на принудительном зацикливании и выходе именно через клавишу <Esc>. Иногда пользователь, забыв выйти из одной программы, выбирает в меню другой пункт. Символы `^C^C` в макроопределении меню прерывают действующую программу. При завершении или прерывании хорошая про-

[1] Так было до появления функций — ловушек ошибок, которые мы рассмотрим далее.

грамма должна восстановить системное окружение. Штатный выход, предусмотренный программистом, обычно предусматривает и восстановление окружения. А как быть при прерывании?

Функция *error*

Прерывание программы, ручное или автоматическое, вызывает стандартный обработчик ошибок AutoCAD — функцию *error*.

Большинство программистов знают, что в своей программе можно, при необходимости, переопределять функцию *error*. Можно-то оно можно, но нередко разработчики делают это неправильно. Во многих пособиях и примерах функция *error*, заменяющая стандартный обработчик ошибок, выглядит ужасающе. Вот несколько примеров. Во всех примерах считается, что в глобальной переменной OLD_ERROR сохранен стандартный обработчик, т. е. где-то производится присваивание

```
(setq OLD_ERROR *error*)
```

Листинг 10.57. Функция *error*. Пример 1

```
(defun *error* (msg)
(princ)
)
```

Это (листинг 10.57) самый бесполезный вариант. Функция просто подавляет стандартное сообщение, а пользователь и не знает об ошибке.

Листинг 10.58. Функция *error*. Пример 2

```
(defun *error* (msg)
  (if OLD_ERROR (setq *error* OLD_ERROR))
  (princ)
)
```

Второй вариант (листинг 10.58) лучше. В нем восстанавливается предыдущий обработчик, но сообщение об ошибке подавляется. Обычно в таком виде функция *error* включается программистами фирмы Autodesk внутрь функции ai_abort. Само сообщение, да еще в форматированном виде, выводится с помощью диалогового окна функции alert.

Листинг 10.59. Функция *error*. Пример 3

```
(defun *error* (msg)
  (print msg)
  (setq *error* OLD_ERROR)
  (princ)
)
```

Совсем бесполезный вариант (листинг 10.59). Функция не делает ничего своего, но сама может стать источником ошибки. Печатается стандартное сообщение и якобы

восстанавливается предыдущее определение. Но даже не сделана проверка переменной OLD_ERROR, и если она не установлена, то функция *error* будет сброшена в nil, а это, как ни странно, восстанавливает стандартный обработчик ошибок.

Как писать функцию *error*

Многие программисты переопределяют эту функцию по инерции (все делают, и я сделаю), не задумываясь о ее применении. Очень редко встречается прямой вызов функции *error* в программе:

```
(*error*
 (strcat "\nФайл" \n" file_name "\n не найден.")
)
```

В этом случае в функцию *error* передается собственное информативное сообщение об ошибке.

Но применить эту функцию можно и с большей пользой. Приведем схему переопределения *error* прямо внутри другой функции, в которой в цикле выполняются некоторые операции, в случае прерывания которых непременно нужно завершить команду (неважно, какую) и удалить временный примитив (листинг 10.60).

Листинг 10.60. Функция *error*. Пример 4

```
(defun ПРИМЕР_ERROR ( / tmp_ent old_local_error)
 (if (ПРОВЕРКА_ЧЕГО_ТО)
  (progn
    (setq old_local_error *error*)
;;; Переопределяем *error* прямо внутри функции, да еще не вначале,
;;; как обычно делается, а внутри условной ветви
    (defun *error* (msg)
     ;; Завершаем команду
      (command "")
      ;; Удаляем временный примитив
      (if tmp_ent (entdel tmp_ent))
      ;; Восстанавливаем прежний обработчик ошибок
      (setq *error* old_local_error)
      (princ)
   )
    (if (ЕЩЕ_ПРОВЕРКА)
      (progn
       (ПОЛЕЗНАЯ_РАБОТА)
;|
Действия, во время которых вызывается команда, которую нужно завершить,
и создается примитив TMP_ENT, который нужно непременно удалить,
 если программа будет прервана пользователем
Во время выполнения этого куска функции пользователь может нажать ESC
|;
      )
   )
```

```
    (setq *error* old_local_error)
   )
  )
)
```

Начиная с системы AutoCAD 2000, функция `*error*` может объявляться локальной. Вернее, объявлять-то локальной ее можно было и раньше, но теперь глобальный обработчик ошибок сам восстанавливается. Предыдущий пример можно упростить (листинг 10.61).

Листинг 10.61. Функция *error*. Пример 5

```
(defun ПРИМЕР_ERROR ( / tmp_ent *error*)
 (if (ПРОВЕРКА_ЧЕГО_ТО)
  (progn
    (defun *error* (msg)
      (command "")
      (if tmp_ent (entdel tmp_ent))
      (princ)
   )
    (if (ЕЩЕ_ПРОВЕРКА) (ПОЛЕЗНАЯ_РАБОТА))
   )
 )
)
```

В системе ruCAD мы будем иногда применять подобные ухищрения, но в обычных программах у нас будет использоваться собственный типовой обработчик ошибок.

Мы сразу предусматриваем два применения для функции `*error*` — типовой обработчик, применяющийся в большинстве программ, и локальные обработчики ошибок со специальными алгоритмами. Пример специального обработчика ошибок мы только что привели. С типовым обработчиком ошибок следует разобраться когда и зачем его требуется применять, в соответствии с этим и реализовывать.

Удобно переопределять и восстанавливать типовой обработчик ошибок в типовых функциях начала и завершения программ. Изменение кода в одном месте будет приводить к изменению поведения всех программ.

Функции начала и завершения приложений

Что должно происходить в начале и конце приложения?

Во-первых, каждую программу, как мы уже указывали, необходимо корректно начинать и завершать. При выходе из программы любым способом все системные переменные, измененные во время работы программы (для ее нужд, а не в виде результата работы), должны восстанавливаться в исходное состояние. Очевидно, при входе в программу их необходимо запомнить.

Часто, если программа ничего особенного не делает, без таких блоков можно и обойтись. Зачем лишние сложности? Например, программы "перелистывания" слоев или изменения свойств текста не нуждаются ни в сохранении окружения, ни в вос-

становлении. А любая "рисовальная" программа, да еще с интерактивным вводом, требует и восстановления окружения, и переопределения `*error*`.

Во-вторых, надо учитывать инстинктивное желание пользователей отменить нажатием клавиши <Esc> только что сделанную работу, или прервать выполнение программы. Функция `*error*` реагирует на прерывание, не разбираясь в причинах. Но причины могут быть и уважительные.

Разберемся сначала со вторым вопросом. "Продвинутые" обработчики ошибок, кроме вывода сообщений и восстановления переменных, позволяют задать возможность отмены изменений, внесенных программой. Примером является библиотека Express Tools. Сейчас в Express Tools используются функции `acet-error-init` и `acet-error-restore`, определенные в файле acetutil.fas. Применять эти функции (и другие 45 "закрытых") можно, но делать ставку на них нельзя, т. к. библиотека по умолчанию не устанавливается, хотя доступна для скачивания и входит в комплект системы AutoCAD 2004. Ради интереса познакомимся, как работают две упомянутых функции.

`(acet-error-init args)` — инициализирует обработчик ошибок.

Функции передается один аргумент, но в виде списка `args`, содержащего три элемента:

- список пар имен системных переменных и их новых значений;
- флаг, указывающий на способ использования команды UNDO:
  - `nil` — не использовать откат;
  - 0 — откат используется и ставятся метки начала и конца отката, но команда UNDO не вызывается при ошибке;
  - 1 — используется откат и команда UNDO при ошибке;
- функция, дополнительно вызываемая при обработке ошибки или `nil`, если не требуется дополнительной обработки.

Пример:

```
(acet-error-init '(("CMDECHO" 0) 1 (if ename (redraw ename 4))))
```

Функция `acet-error-restore`, не имеющая аргументов, восстанавливает обработчик ошибок и значения, сохраненные при вызове первой функции `acet-error-init`.

Примеры использования этих функций можно посмотреть в исходных текстах программ библиотеки.

Совет

Каждый разработчик должен изучать исходники библиотеки Express Tools! Это лучший из общедоступных примеров разработки очень надежных функций на языке LISP.

Мы несколько лет использовали подобный обработчик ошибок, дополненный запросом у пользователя подтверждения отмены внесенных изменений или отказа от них. Опыт показал, что квалифицированные пользователи, нажимающие клавишу <Esc> сознательно, лишь *иногда* отменяют изменения, внесенные программой. Они умеют использовать команду UNDO и предпочитают пользоваться именно ей, контролируя процесс отката. Пользователи категории "тетка", прерывающие программу

в полубессознательном состоянии, вообще всегда отказываются от отката, и правильно делают — неизвестно ведь, "чего и сколько" удалится. Например, если *начало* и *концовка* неразумным программистом были вставлены в команду вызова меню, то может отмениться вся работа, проделанная после выбора такого пункта меню.

В результате, при разработке системы ruCAD мы решили отказаться от навязывания услуг по *полуавтоматическому* откату изменений в блоках начала и завершения программ, но применять его, по необходимости, в требуемых местах. Как удобнее предусматривать откат мы еще поговорим.

Вернемся к более простому вопросу — что сохранять и восстанавливать. Обычно в начале программ мы изменяем следующие переменные (запоминая их прежние значения):

- CMDECHO — 0, чтобы не пугать "теток" лишними сообщениями;
- BLIPMODE — 0, чтобы не засорять экран "крестиками";
- PICKBOX — просто запоминаем, потому что в программе размер прицела может изменяться;
- PLINEWID — запоминаем, потому что в программах ширина полилиний постоянно меняется;
- HIGHLIGHT — просто запоминаем, на всякий случай;
- ORTHOMODE — просто запоминаем;
- CELTYPE — запоминаем, потому что в программах тип линий часто изменяется.

В особых случаях можно запоминать и другие переменные.

Типовая функция начала программы приведена в листинге 10.62.

Листинг 10.62. Функция `ru-app-begin`

```
(defun ru-app-begin ()
(gc); проводим сборку мусора
;; если не включен режим отладки
;; Инициализируем обработчик ошибок
  (if (not *ru_msg_debug*)(ru-error-init))
)
```

Типовая функция завершения программы приведена в листинге 10.63.

Листинг 10.63. Функция `ru-app-end`

```
(defun ru-app-end ()
  (if (not *ru_msg_debug*)
    (ru-error-restore);;восстанавливаем окружение
 )
)
```

Функции `ru-error-init` и `ru-error-restore` мы рассмотрим позже. Функция досрочного выхода, о которой мы также говорили, приведена в листинге 10.64.

Листинг 10.64. Функция `ru-app-exit`

```
(defun ru-app-exit (message)
 (ru-msg-alert message)
 (ru-app-end)
 (exit)
)
```

Типовой обработчик ошибок ruCAD

Теперь разработаем свой обработчик ошибок (листинг 10.65).

Листинг 10.65. Функция `ru-error`

```
(defun ru-error (msg)
;; Эта функция заменяет error и ей будет передано сообщение об ошибке
 (if (member msg
        '("console break"
          "Function cancelled" "Функция отменена"
          "quit / exit abort" "завершить / выйти прервать"
        )
      );_ end of member
    (princ "\nПрерывание команды ruCAD! ")
;; В более сложных случаях выдаем дополнительную информацию
    (princ
      (strcat "\ERRNO # "
              (itoa (getvar "ERRNO"))
              ": "
              msg
             "\n"
     )
   )
 )
 ;; Завершаем все активные команды.
  (while (/= (getvar "CMDACTIVE") 0) (command nil))
 ;; Восстанавливаем окружение
  (ru-error-restore)
  (princ)
)
```

А теперь самое интересное — сохранение и восстановление переменных (листинги 10.66 и 10.67).

Листинг 10.66. Функция `ru-error-init`

```
(defun ru-error-init ()
;| Запоминаем прежний обработчик ошибок в глобальной переменной и переопределяем на
свой |;
  (setq *ru_old_error* *error* *error* ru-error)
```

```
;; Устанавливаем новые значения системных переменных, запоминая текущие значения
  (ru-error-save-sysvars '(
                     ("CMDECHO" 0)("BLIPMODE" 0)("OSMODE")
                     ("PICKBOX") ("PLINEWID")("HIGHLIGHT")
                     ("ORTHOMODE")("CELTYPE")
                    )
 )
)
```

Обратите внимание — некоторые подсписки не парные! Передается только имя переменной, при этом новое значение не будет установлено, а только запомнится текущее значение, на случай, если в программе эти переменные будут меняться.

Листинг 10.67. Функция `ru-error-restore`

```
(defun ru-error-restore ()
  (ru-error-restore-sysvars)
  (if *ru_old_error* (setq *error* *ru_old_error*))
)
```

Функция сохранения системных переменных приведена в листинге 10.68.

Листинг 10.68. Функция `ru-error-save-sysvars`

```
(defun ru-error-save-sysvars (list_sysvar_lists / var_name var_new_value
                                                  var_old_value)
 ;; Сохраняет текущее значение системных переменных
 ;; и устанавливает новые значения, если они заданы
 ;; Список сохраняется в глобальной *ru_sysvars*
;; В каждом подсписке (ПЕРЕМЕННАЯ НОВОЕ_ЗНАЧЕНИЕ)
  (foreach sysvar_sub_list list_sysvar_lists
   ;; Имя переменной
    (setq var_name (car sysvar_sub_list))
   ;; Выясняем текущее значение
    (setq var_old_value (getvar var_name))
   ;; Выясняем новое значение, которое надо установить
    (setq var_new_value (cadr sysvar_sub_list))
   ;; Мы добавляем в список глобальных! При повторном
   ;; применении в списке будет история присваиваний
    (setq *ru_sysvars_list*
       (cons (list var_name var_old_value)
            *ru_sysvars_list*
       )
   )
   ;; Если в подсписке имеется новое значение
    (if var_new_value
     ;; Устанавливается заданное значение
      (setvar var_name var_new_value)
   )
 )
)
```

Настоящий LISP-программист, которому красота функции важнее, чем понимание ее работы посторонними лицами, непременно написал бы так:

```
(foreach sysvar_sub_list list_sysvar_lists
 (set '*ru_sysvars_list*
   (cons (list (car sysvar_sub_list) (getvar (car sysvar_sub_list)))
      (eval '*ru_sysvars_list*)))
 (if (cadr sysvar_sub_list)
   (setvar (car sysvar_sub_list) (cadr sysvar_sub_list)))
)
```

Листинг 10.69. Функция `ru-error-restore-sysvars`

```
(defun ru-error-restore-sysvars (/ var_name var_stored_value)
  (foreach sysvar_sub_list *ru_sysvars_list*
    (setq var_name (car sysvar_sub_list))
   ;; Выясняем текущее значение
   ;; Выясняем новое значение, которое надо установить
    (setq var_stored_value (cadr sysvar_sub_list))
    (setvar var_name var_stored_value)
 );_ end of foreach
 ;; А можно так:
;;  (foreach sysvar_sub_list *ru_sysvars_list*
;;    (setvar (car sysvar_sub_list) (cadr sysvar_sub_list))
;; )
  (setq *ru_sysvars_list* nil)
)
```

Проблемы с откатом

Мы уже отказались от массового использования отката при ошибках прерывания, но это не значит, что не надо заботиться об удобстве отмены операций пользователем. Для того чтобы пользователь мог отменить целую группу команд, в программу вставляют метки начала и конца группы. Проблема бывает с переменной CMDECHO. Обычно на время работы программы эта переменная устанавливается в ноль, чтобы не засорять экран. Предыдущее состояние CMDECHO, как и других переменных, должно быть восстановлено. Если до работы программы CMDECHO была 1, то и после выхода должна остаться 1. Когда откат группами в программе не предусмотрен, ничего особенного не происходит. Но программа во время работы может выполнить очень много действий и желательно дать возможность пользователю отменять их логическими группами, а может и все сразу. В число отменяемых действий войдет и изменение переменной CMDECHO. Для этого марка начала группы должна быть установлена до сброса CMDECHO, однако в этом случае отобразится эхо самой команды. Рассмотрим примеры. Вариант с установкой метки отката командой приведен в листинге 10.70.

Листинг 10.70. Установка меток группы отката командой

```
(defun c:cmd_undo ( / pt1 pt2)
;;; Имитируем установку CMDECHO вне нашей программы
(setvar "cmdecho" 1)
```

```
;;; Устанавливаем метку начала отката. Текст команды явно будет
;;; выведен на экран
(command "_.UNDO" "_begin") ;; это будет видно
(setvar "cmdecho" 0);; Отключаем эхо команд
(while
(setq pt1 (getpoint "\nПервая точка <Выход>: "))
(initget 1)
(setq pt2 (getpoint pt1 "\nВторая точка: "))
(command "_.LINE" pt1 pt2 "")
)
;;; Устанавливаем метку конца группы
(command "_.UNDO" "_end")
(setvar "cmdecho" 1)
(princ)
)
```

Проверим работу программы.

```
Command: cmd_undo
_.UNDO Enter the number of operations to undo or
[Auto/Control/BEgin/End/Mark/Back] <1>: _begin
Command:
Первая точка <Выход>:
Вторая точка:
Первая точка <Выход>:
Вторая точка:
```

Мы нарисовали два отрезка (могли и две тысячи). Если пользователь теперь щелкнет по кнопке отката на панели инструментов (макрос `_UNDO _group`), сотрутся все нарисованные отрезки. Работу программы портит вывод сообщения команды UNDO.

Теперь напишем вариант с использованием методов ActiveX (листинг 10.71).

Листинг 10.71. Установка меток группы отката методами ActiveX

```
(defun c:ax_undo ( / pt1 pt2 doc_obj mspace line)
(setvar "cmdecho" 1)
(setq doc_obj (vla-get-activedocument
                          (vlax-get-acad-object)))
(setq mspace (vla-get-modelspace doc_obj))
(vla-startundomark doc_obj)
(setvar "cmdecho" 0)
(while
(setq pt1 (getpoint "\nПервая точка <Выход>: "))
(initget 1)
(setq pt2 (getpoint pt1 "\nВторая точка: "))
(setq line (vla-addline mspace
               (vlax-3d-point pt1)(vlax-3d-point pt2)))
)
(vla-endundomark doc_obj)
(setvar "cmdecho" 1)
(princ)
)
```

В этом примере метки начала и конца группы отката мы поставили с помощью методов StartUndoMark и EndUndoMark. Метод мы могли применить только к объекту, поэтому пришлось получить объект самой системы AutoCAD, у него затребовать активный документ и метки ставить в активном документе. А раз уж мы забрались в объектную модель, то и отрезки рисовали объектными методами, хотя могли применить и функцию command. В этом случае метки можно было поставить без создания дополнительных переменных:

```
(vla-StartUndoMark (vla-Get-ActiveDocument (vlax-Get-Acad-Object)))
(vla-EndUndoMark (vla-Get-ActiveDocument (vlax-Get-Acad-Object)))
```

Ошибки свойств

При работе через ActiveX возможны попытки обращения к свойствам, отсутствующим у объекта, или изменения свойств, не подлежащих редактированию.

Листинг 10.72. Получение длины объекта. Вариант 1

```
(defun length1 ( / ent obj)
(while (setq ent (entsel "\nУкажи объект для проверки длины: "))
  (progn
    (setq obj (vlax-ename->vla-object (car ent)))
    (princ
      (strcat "\nДлина: " (rtos (vla-get-Length obj)))
   )
 )
)
)
Command: (length1)
Укажи объект для проверки длины:
Длина: 189
Укажи объект для проверки длины:; error: ActiveX Server returned the error:
unknown name: Length
```

После первого запроса мы указали на отрезок, после второго — на полилинию. Ну никак пользователь не может предположить, что у полилинии "нет длины". Программист мог бы это знать, мог бы ограничить выбор объекта по типу. А мог бы и проверить доступность свойства (листинг 10.73).

Листинг 10.73. Получение длины объекта. Вариант 2

```
(defun length2 ( / ent obj)
 (while (setq ent (entsel "\nУкажи объект для проверки длины: "))
  (progn
    (setq obj (vlax-ename->vla-object (car ent)))
    (if (vlax-property-available-p obj 'Length)
      (princ
        (strcat "\nДлина: " (rtos (vla-get-Length obj)))
     )
```

```
      (princ "\nУ этого объекта нет свойства 'LENGTH'")
   )
 )
)
)
```

Примечание

Интересно, а зачем придумано две формы получения и установки свойств, например:

`(vlax-put-property obj 'Color 7)` **и** `(vla-put-color obj 7)`**?**

Смысл отделения имени свойства от функции (первый вариант), видимо, в том, что появляется возможность обработки ассоциированных списков имен свойств и их значений.

Для проверки редактируемости свойств объектов служит необязательный ключ функции `vlax-property-available-p`. В этом случае вызов может выглядеть примерно так: `(vlax-property-available-p obj 'Length T)`.

Ловушки для ошибок

Функция `*error*` что-то делает (или не делает) уже *после* того, как произошла фатальная для программы ошибка, например "обезьяна с гранатой" на момент вопроса "Да/Нет" нажала клавишу <Esc>, воображая, что это соответствует ответу "Нет". Можно предотвратить и такую ситуацию. Вот наша рабочая функция запроса "Да/Нет" с ответом по умолчанию "Нет" (листинг 10.74). Эта функция будет применяться тогда, когда выводить диалоговое окно нецелесообразно или невозможно.

Листинг 10.74. Функция `ru-dlg-yes-cml`

```
(defun ru-dlg-yes-cml (msg / result)
 (setq msg (strcat "\n" msg " [Да/Нет]<Да>? : "))
  (while
    (vl-catch-all-error-p
      (progn
       (initget  "Да Нет")
       (setq result (vl-catch-all-apply
                              'getkword
                               (list msg)))
     )
   )
 )
   (not (= result "Нет"))
)
```

При работе этой функции[1] нажатие клавиши <Esc> не приводит к прерыванию программы.

```
Command: (ru-dlg-yes-cml "Продолжим изучение LISP")
Продолжим изучение LISP [Да/Нет]<Да>? : *Cancel*
```

[1] Обратите внимание, что опции `[Да/Нет]`, записанные в квадратных скобках, появляются и в контекстном меню. Правда, в контекстных меню лучше избегать русских опций, потому что AutoCAD (особенно русский!) на них часто сбивается (теряет первый символ).

```
Продолжим изучение LISP [Да/Нет]<Да>? : No
Invalid option keyword.
Продолжим изучение LISP [Да/Нет]<Да>? : Нажали Enter
T
```

Избавиться от вопросов удается только при вводе одного из правильных ответов или при пустом ответе, трактуемом как "Да". В этом примере работают сразу две функции-ловушки:

- `vl-catch-all-error-p`
- `vl-catch-all-apply`

Функция `vl-catch-all-apply` получает два аргумента: ссылку на имя функции и список параметров этой функции. В примере выделенное выражение заменяет традиционный вызов

```
(setq result (getkword "\nПродолжим изучение LISP? [Да/Нет]: "))
```

Первым аргументом функции `vl-catch-all-apply` явилось имя функции `getkword`, заданное с апострофом, вторым — список аргументов, передаваемых функции `getkword` (в примере это список из одного аргумента). Функция `vl-catch-all-apply` вернет результат функции `getkword`, но, если во время выполнения `getkword` произойдет ошибка (в нашем случае почти наверняка это будет нажатие клавиши <Esc>), то `vl-catch-all-apply` вернет специальный объект типа `VL-CATCH-ALL-APPLY-ERROR` с характеристиками ошибки. Как воспользоваться этим объектом и надо ли устраивать его анализ, решается в контексте применения функции. В данном примере нас не интересует, какая ошибка произошла, нам нужно обязательно получить безошибочный результат, т. е. одно из допустимых ключевых слов.

Для этого мы в цикле функции `while` проверяем результат с помощью функции `vl-catch-all-error-p`, передавая ей в качестве аргумента результат функции `progn` (в данном случае это переменная `result`). Функция `vl-catch-all-error-p` проверяет, является ли значение переменной `result` объектом типа `VL-CATCH-ALL-APPLY-ERROR`. Если тип значения переменной `result` не `VL-CATCH-ALL-APPLY-ERROR`, функция `vl-catch-all-error-p` вернет `nil`, произойдет выход из цикла `while` и нам остается только определить, какое именно значение переменной `result` — "Да" или "Нет" (вернее, мы проверяем условие "не Нет", т. е. "Да"), т. к. иного результата уже быть не может, в том числе невозможен "вылет" программы из-за нажатия клавиши <Esc>.

Если бы нас интересовало, *какая именно* ошибка отловлена функцией `vl-catch-all-apply`, мы могли извлечь сообщение об ошибке из объекта `VL-CATCH-ALL-APPLY-ERROR` с помощью функции `vl-catch-all-error-message`:

```
(if (= (type result) 'VL-CATCH-ALL-APPLY-ERROR)
 (princ (vl-catch-all-error-message result)
)
```

Идеология функций `vl-catch-xxx` кажется чуждой LISP, но появилась не случайно. Связано это с активным использованием механизмов ActiveX, на которых мы подробнее задержимся в последующих главах. Подобные конструкции работают аналогично блокам `try ... except` в Delphi или `Try-Catch exception` в C++. Там они когда-то тоже казались непривычными, но программисты быстро оценили преимущества таких решений. Да и для "чистого" языка LISP набор этих функций открыл новые возможности. Например, в версиях AutoCAD R10 — R12 предотвратить сбой программы при нажатии клавиши <Esc> было просто невозможно.

В последующих главах мы неоднократно будем применять конструкции типа:

```
(if (setq com_server (vlax-get-or-create-object "MyServer.MyObject"))
  (работа_с_com_server)
  (princ "\nНе удалось запустить MyServer.MyObject")
)
```

Аналогичный результат, даже более информативный, можно было бы получить с использованием конструкции

```
(cond
 ((vl-catch-all-error-p
   (setq com_server (vl-catch-all-apply 'vlax-create-object
    ("MyServer.MyObject")))
 )
  (vl-exit-with-error (strcat "\nОшибка: "
    (vl-catch-all-error-message com_server)))
)
 (T
  (работа_с_com_server)
)
)
```

Какую структуру лучше применять — зависит от контекста программы, а еще больше от личных пристрастий автора. Мы пока придерживаемся традиционной консервативной схемы. Возможно, жизнь за это нас еще накажет.

Использование ловушек ошибок позволяет избежать многих сбойных ситуаций. Например, легко предотвращается "смертельная" ошибка деления на ноль:

```
(setq result (vl-catch-all-apply '/ '(x y)))
```

Если значение `y` по каким-либо причинам окажется равным нулю, результатом будет:

```
#<%catch-all-apply-error%>
```

Такую ситуацию можно обработать:

```
(vl-catch-all-error-message result)
```

и получить сообщение `"divide by zero"` без фатального прерывания программы.

"Ловля блох" в ActiveX

Иногда ошибки невозможно отследить, предотвратить и проинформировать об них пользователя без специальных функций. При использовании ActiveX ошибка может произойти по разным причинам, но внятного сообщения пользователь не получит, если не считать туманного:

```
Error: ActiveX error: No description provided.
```

При работе с ActiveX есть два ключевых момента.

Во-первых, мы всегда должны получить какой-то объект, чтобы прочитать или изменить значения нужных нам свойств. Увидеть свойства "главного" для нас объекта

AutoCAD можно только получив указатель на него с помощью функции `vlax-Get-Acad-Object`. Добраться до объектов, спрятанных глубоко в иерархии, можно только проходя по цепочке всех их владельцев. И на всем пути нас подстерегают возможные ошибки.

Во-вторых, многие функции разрабатываются с возвратом исключений. Это связано с использованием в COM для результатов типа `HResult`. Подробности мы узнаем при разработке собственных COM-серверов, а пока будем иметь в виду, что функции могут возвращать полезный результат или исключение при какой-то неудаче (исключительной ситуации). Функция `*error*` на такие ошибки не реагирует.

Отлов ошибок при отладке

Несмотря на то, что мы уже столько знаем о предотвращении ошибок, нам их не избежать. Ошибки будут всегда! И их нужно искать. За рубежом принято ошибки деликатно называть "маленькими жучками", наши люди этих "жучков" не стесняются называть по имени, а процесс их поиска — "ловля блох". Ну, "скифы мы".

Фредерик П. Брукс, автор знаменитой книги "Мифический человеко-месяц или как создаются программные системы"[1], справедливо пишет: "Разработка грандиозных идей — это удовольствие, а поиск паршивых маленьких "жучков" — это всего лишь работа". Грандиозных идей мы уже наразрабатывали, но все они могут сорваться, если не научимся бороться с "малышами".

В не столь отдаленные времена единственным способом отладки программ в AutoLISP было отслеживание ошибок путем вставки в текст программы различных сообщений и вывода на экран значений переменных. AutoCAD выводил на экран кусок исходного текста программы, в случае прерывания по ошибке. В системе AutoCAD 2000 и старше текст программы уже не выводится, а дается очень лаконичное сообщение.

Использование отладочных сообщений

Интегрированная среда Visual LISP предоставляет дополнительные возможности по отладке программ, но и вставка информационных сообщений не потеряла актуальность. Причины заключаются в том, что не всегда может быть доступен исходный текст программы. Вывод значений переменных удобно делать функциями `print` и `princ`, выводящими на экран данные любых типов. Отладочные сообщения вставлены заранее в текст программы, но выводятся только при включении режима отладки вне отслеживаемой программы. Такой режим можно рекомендовать пользователю для выявления сбойных ситуаций перед обращением за технической поддержкой. Протокол работы программы можно направить разработчикам. Для отслеживания логики работы удобно использовать специальные отладочные функции (листинг 10.75).

---

[1] THE MYTHICAL MAN-MONTH (Essays on Software Engineering) ADDISON-WESLEY PUBLISHING COMPANY READING 1975 Copyright (c) 1975 by Addison-Wesley Publishing Company, Inc. Philippines; Copyright 1975 by Addison-Wesley Publishing Company, Inc. Copyright (c) 1972 by Frederick P. Brooks, Jr. OCR, formatting: Jek, Alex Buloichik.

Листинг 10.75. Функция ru-msg-debug

```
(defun ru-msg-debug (n /)
(if *ru_msg_debug*
(princ (strcat "\n>>>> Точка отладки N ** " (itoa n) " <<<< \n")))
)
```

Вставляя вызов этой функции с различными параметрами в тело программы, можно найти точки, в которых программа ломается. Сообщение выводится, если глобальная переменная *ru_msg_debug* не nil. Включение или выключение этой переменной производится в меню **Профи** вот таким макросом:

```
[~Режим отладки]^C^C^P(ru-var-toggle-var "*ru_msg_debug*")
```

Функция ru-var-toggle-var меняет значение переменной на противоположное, а также ставит или снимает пометку пункта меню. При включенном режиме отладки будут выводиться все отладочные сообщения. Конечно, можно установить режим отладки и вручную:

```
(setq *ru_msg_debug* T)
```

"Улучшим" нашу тренировочную функцию (листинг 10.76) и попробуем ее выполнить.

Листинг 10.76. Получение длины объекта. Вариант 3

```
(defun length3 ( / ent obj)
(while (setq ent (entsel "\nУкажи объект для проверки длины: "))
  (progn
    (ru-msg-debug 1)
    (setq obj (vlax-ename->vla-object (car ent)))
    (ru-msg-debug 2)
    (cond
      ((vlax-property-available-p obj 'Length)
        (ru-msg-debug 3)
        (princ (strcat "\nДлина: "  (vla-get-Length obj)))
        (ru-msg-debug 4)
     )
      (T (princ "\nУ этого объекта нет свойства 'LENGTH'")
     )
   )
 )
)
)
Command: (length3)
Укажи объект для проверки длины:; error: bad argument type: stringp 277.839
```

Это явная и предсказуемая ошибка. Мы пытаемся передать функции strcat число вместо строки.

При включенном режиме отладки многострадальная функция выдаст сообщения:

```
Укажи объект для проверки длины:
>>>> Точка отладки N ** 1 <<<<
>>>> Точка отладки N ** 2 <<<<
>>>> Точка отладки N ** 3 <<<<
; error: bad argument type: stringp 277.839
```

Мы видим, что сбой произошел между точками 3 и 4. Конечно, в этом примере все ясно без отладки, но бывают сложные варианты. Случается, что у пользователя, находящегося где-то далеко, постоянно происходят сбои, а у автора все нормально. Для того чтобы воспроизвести ситуацию, попросите включить режим отладки через меню и выслать протокол работы, скопированный с текстового экрана AutoCAD.

Примечание

Следствию очень поможет и просмотр файла acadinfo.txt, сгенерированного стандартной программой acadinfo.lsp. Переименовав acadinfo.txt в LSP-файл и загрузив его, можно воссоздать в своем рисунке такое же окружение, как и у пользователя.

Возможно использование и более изощренных сообщений (листинг 10.77).

Листинг 10.77. Функция *PRINT*

```
(defun *PRINT* (msg var n)
 ;; печать в режиме отладки сообщения MSG, значения переменной VAR
 ;; с задержкой на N секунд или до нажатия клавиши
  (if *ru_msg_debug*
    (progn
      (princ "\n** ОТЛАДОЧНОЕ ПРЕРЫВАНИЕ **\n")
      (princ msg)
      (princ "\nПеременная: ")
      (princ var)
      (princ "\nЗначение: ")
      (princ (eval var))
      (princ "\nТип: ")
      (princ (type (eval var)))
      (if n
       (ru-time-wait n)
       (getstring "\nЖми пробел для продолжения...")
     )
      (princ "\n**     ПРОДОЛЖЕНИЕ       **\n")
   )
 )  (princ)
)
```

Немного переделаем тестовую функцию (листинг 10.78) и запустим ее на выполнение.

Листинг 10.78. Получение длины объекта. Вариант 4

```
(defun length4 ( / ent obj l)
(while (setq ent (entsel "\nУкажи объект для проверки длины: "))
```

```
  (progn
    (ru-msg-debug 1)
    (setq obj (vlax-ename->vla-object (car ent)))
    (ru-msg-debug 2)
    (cond ((vlax-property-available-p obj 'Length)
    (ru-msg-debug 3)
      (setq l (vla-get-Length obj))
      (*PRINT* "Точка отладки такая-то..." 'l nil)
      (princ
        (strcat "\nДлина: "  L)
     )
    (ru-msg-debug 4)
     )
      (T (princ "\nУ этого объекта нет свойства 'LENGTH'")
     )
   )
 )
)
)
Укажи объект для проверки длины:
ru-msg-debug: >>>> Точка отладки N ** 1 <<<<
ru-msg-debug: >>>> Точка отладки N ** 2 <<<<
ru-msg-debug: >>>> Точка отладки N ** 3 <<<<
** ОТЛАДОЧНОЕ ПРЕРЫВАНИЕ **
Точка отладки такая-то...
Переменная: L
Значение: 37965.5
Тип: REAL
Жми пробел для продолжения...
**     ПРОДОЛЖЕНИЕ       **
; error: bad argument type: stringp 195.134
```

Теперь все становится совершенно ясно.

Встроенные средства отладки Visual LISP

При наличии исходных текстов подобные ошибки можно находить гораздо быстрее в IDE Visual LISP. Описывать, как это делается, еще и в этой книге мы не будем. Напомним только, что отладочные возможности Visual LISP сосредоточены в меню **Debug** (Отладка) и одноименной панели инструментов.

Как не стать параноиком

Узнав столько жутких подробностей об ошибках, легко поддаться панике и начать насыщать все функции всевозможными проверками. Здесь важно вовремя остановиться. Если мы начнем, например, *всегда* проверять типы переданных данных, ничего хорошего не получится. Скорее всего, мы наделаем ошибок при проверках на наличие ошибок. Ввод пользователя проверять надо, а проверять, правильные ли параметры передали мы сами в свои же функции, как правило, не следует. Иначе потом возникнет подозрение, что мы неправильно выполнили проверку. В результате мы никогда не закончим разработку своей системы. А для того чтобы свести к

минимуму ошибки, связанные с передачей данных неверных типов или выходящих за пределы рабочего диапазона, при разработке новых функций, нужно в описании функции[1] *всегда* подробно указывать, какие данные может принимать функция и *абсолютно все* варианты возможных возвратов.

Особое внимание следует уделить описанию исключений и возврату при нештатных ситуациях, если таковые предусмотрены. Предусматривать такие "нештатные возвраты", в первую очередь, имеет смысл для функций, по сути, являющихся полуавтономными программами, т. е. выполняющих ряд действий в активном взаимодействии с пользователем или с глобальным окружением. Так как и действия пользователя, и глобальное окружение подконтрольны нам лишь отчасти, то в случае, если для дальнейшей работы нам будут необходимы результаты действия вызываемой функции, мы далеко не всегда можем быть уверены в том, что все прошло так, как ожидалось. Такой "нештатный возврат" позволяет определить, были ли действия успешно выполнены, в чем причина неудачи, и принять решение о дальнейшем поведении всей программы.

Создание приложений Visual LISP

С помощью Visual LISP можно писать не только отдельные программы, но и целые приложения. А что это такое — "приложение"? Мы-то что делаем — просто "большую кучу" программ и библиотек функций, много приложений или одно большое приложение по имени ruCAD? С "точки зрения Visual LISP" приложением считается отдельный исполняемый файл, предназначенный для использования конечными пользователями, при этом имеется в виду, что собран этот файл с помощью Visual LISP. В нашей системе просто напрашивается создание нескольких "приложений" — библиотек функций и множества приложений-программ, предназначенных для решения конкретных задач (это то, что пользователи будут выбирать из меню).

Для того чтобы с помощью Visual LISP собрать приложение, необходимо сначала создать проект. Возможны такие варианты:

- из меню Visual LISP **Project > New** — создать файл проекта с расширением prj (именуется "VLisp project file");
- из меню Visual LISP **File > Make Application > Make Application Wizard** — создать с помощью мастера файл с расширением prv (именуется "Visual LISP make file").

Большую путаницу вносят неточности в терминологии, предложенной фирмой Autodesk. При создании проекта по первому варианту в меню присутствует термин *Visual LISP Project*, в диалоговом окне выбора файла — *VL project files*, окно свойств проекта называется *Project Properties*, построение проекта уже называется в меню *Build Project FAS*, а сам проект имеет заголовок *VLisp project file*. Файл исходного текста проекта имеет расширение prj и в справочной системе называется *Project definition*, а *построенный* файл имеет расширение fas и, по версии справочной системы, называется *Compiled AutoLISP code*.

При создании приложения с помощью пункта меню *Make Application Wizard* все названия включают слово *Application*, исходный код с расширением prv имеет заголо-

[1] Увы, когда дело доходит до документирования, благие намерения обычно забываются.

вок *Visual LISP make file*, а такой тип файлов, по версии справочной системы, называется *Make File*, но предназначен для построения (Build) *VLX application*. Собранный файл с расширением vlx в справке сначала называется *Standalone Application*, а затем *Visual LISP executable*.

Возможно, нам мешают излишние в данном случае знания о том, что такое в программировании проекты, компиляция, Standalone application, процессы Build и Make, но давайте все-таки определимся с терминологией хотя бы для этой книги. Впредь мы будем называть:

- LSP-файлы — исходными текстами (программ или библиотек функций);
- FAS-файлы — FAS-приложениями (программами или библиотеками функций), откомпилированными из исходных текстов в соответствии с параметрами проектов;
- PRJ-файлы — проектами FAS-приложений;
- VLX-файлы — VLX-приложениями (типа Simple или Expert);
- PRV-файлы — проектами VLX-приложений из FAS-файлов и ресурсов (DCL- и TXT-файлов).

Создание FAS-приложения

При создании приложения из меню Visual LISP **Project > New** можно:

- сформировать список LSP-файлов, включаемый в проект;
- задать папки для FAS-файлов и временных файлов;
- задать режим слияния файлов — делать один FAS-файл для всех исходных или отдельные FAS-файлы;
- установить объем сообщений о процессе компиляции;
- отредактировать глобальные определения.

Файл проекта "VLisp project file" физически является LISP-программой, сохраненной в файле с расширением prj. Такой файл можно написать и вручную. PRJ-файл проекта приведен в листинге 10.79.

Листинг 10.79. Пример PRJ-файла

```
;;; VLisp project file [V2.0] ru-lib-all saved to:[C:/.ru/cad/Source/Lisp/LIB]
at:[13.06.03]
(vlisp-project-list
  :name
  ru-lib-all
  :own-list
("ru-add" "ru-ADO" "ru-app" "ru-arc" "ru-circle" "ru-conv" "ru-dirs" ru-doslib"
"ru-ent" "ru-express" "ru-file" "ru-file-dlg" "ru-get"
"ru-ini" "ru-is" "ru-line" "ru-list" "ru-lsp" "ru-lspdata" "ru-ltype"
"ru-match" "Ru-menu" "ru-normal" "ru-pline" "ru-points" "ru-reactors"
"ru-scale" "ru-string" "ru-text" "ru-unit" "ru-user" "ru-vlr" "ru-xml"
 "Ruax-lib" "ru-acad"   )
```

```
  :fas-directory
  "C:\\.ru\\cad\\All Users\\app"
  :tmp-directory
  "C:\\.ru\\cad\\Source\\Lisp\\LIB"
  :project-keys
  (:build (:standard) :merged t :safe-mode t :msglevel 1)
  :context-id
  :autolisp
);_ end of VLISP-PROJECT-LIST

;;; EOF
```

В данном случае мы отчетливо видим список файлов, из которых будет собран проект, и понимаем, что будет собран один FAS-файл. Такое приложение считается *простым*.

Создание проекта дает разработчику дополнительные удобства. IDE Visual LISP позволяет загрузить все файлы проекта в AutoCAD, производить поиск именно в этих файлах, перестраивать проект. А если мы догадаемся, что исходный код проекта вручную редактировать (например, добавлять файлы в список) удобнее, чем через Мастер, то вообще будет проще работать.

Разрабатывая библиотеки для системы ruCAD, мы применяем именно "ручное" редактирование файла проекта. Объясняется это тем, что у нас очень много функций и размещение их в одном каталоге доставляет множество неудобств из-за большого количества файлов. В *главе 9* мы уже писали о размещении групп функций в подкаталогах и о навигации по ним с помощью программы ruLispExplorer. Если все файлы, включаемые в проект, лежат в одном каталоге, IDE Visual LISP самостоятельно находит "новичков", но процесс этот довольно длительный. За сотни "перестроек" проекта это изрядно надоедает, поэтому мы, сформировав первоначальный набор файлов и создав проект автоматически, в дальнейшем будем редактировать его вручную, добавляя строчки с новыми функциями в обычном текстовом редакторе. Фрагмент "настоящего" файла проекта приведен в листинге 10.80.

Листинг 10.80. Фрагмент файла ru-lib-main.prj

```
;;; VLisp project file [V2.0] ru-lib-main saved to:[C:/.ru/cad/Source/Lisp/LIB/FAS]
at:[9/24/03]
(VLISP-PROJECT-LIST
  :NAME
  ru-lib-main
  :OWN-LIST
  (
"3d/ru-3dface-get-perimeter"
"acad/ru-acad-status-bar-button"
"acad/ru-acad-store-app-string"
"acad/ru-acad-ver"
"adc/ru-adc-navigate"
"ado/ru-ado-build-connection-string"
"ado/ru-ado-connect-to-db"
...
```

```
"block/ru-block-insert-scaled-ptask"
"block/ru-block-insert-scaled-ptask-angleask"
"block/ru-block-insert-table"
...
"xml/ru-xml-get-sdata"
"xml/ru-xml-pop-mnu"
"xml/ru-xml-select-macro"
"xml/_ru-xml-tree-select"
"xref/ru-xref-list-with-path"
"xref/ru-xref-path-by-short-name"
  )
  :FAS-DIRECTORY
  "C:/.ru/cad/All Users/app"
  :TMP-DIRECTORY
  "tmp"
  :PROJECT-KEYS
  (:BUILD (:standard) :MERGED T :SAFE-MODE T :MSGLEVEL 1)
  :CONTEXT-ID
  :autolisp
);_ end of VLISP-PROJECT-LIST
;;; EOF
```

Обратите внимание на то, что теперь у нас указаны не просто имена файлов, как в листинге 10.79, а с подкаталогами относительно расположения файла проекта, например "xml/ru-xml-select-macro". Для облегчения навигации внутри большого файла проекта строки с перечнем файлов отсортированы.

Файл проекта можно включить в проект VLX-приложения, что значительно упрощает его подготовку.

Но для того, чтобы откомпилировать LSP-файл в FAS-приложение, проект создавать не обязательно. Это можно сделать с помощью функции `vlisp-compile`. Забегая вперед, скажем, что именно так мы будем автоматически компилировать исходные тексты в функции загрузки приложений.

Создание VLX-приложений

Кроме простых FAS-приложений, Visual LISP позволяет создавать VLX-приложения, в которые, помимо исходных LSP-файлов, могут быть включены DCL-файлы диалоговых окон, FAS-файлы, DVB-файлы[1] (приложения, созданные на VBA) и текстовые файлы. Все включаемые компоненты собираются в единый файл приложения. Здесь главный принцип — "все свое ношу с собой".

При создании приложения с помощью Make Application Wizard можно:

- выбрать один из двух режимов (Simple — включение только LSP-файлов) или (Expert — включение в приложение дополнительных ресурсов);
- задать, будет ли использовать приложение отдельное пространство имен и ActiveX;
- включить в проект дополнительные ресурсы.

[1] Включать-то (в версиях 2000—2002) можно, а вот запускать пока нельзя.

Файл "Visual LISP make file" физически также является LISP-программой, сохраненной в файле с расширением prv. Такой файл можно писать и редактировать вручную (листинг 10.81).

Листинг 10.81. Пример PRV-файла

```
;;; Visual LISP make file [V1.0] ru-lib-all-vlx saved to:[C:/.ru/cad/All Users/app]
at:[13.06.03]
(prv-def (:target . "ru-lib-all-vlx.VLX")
 (:active-x . t)
 (:separate-namespace . t)
 (:protected . t)
 (:load-file-list
   (:lsp "C:/.ru/cad/Source/Lisp/LIB/ru_block_descr.LSP")
);_ end of :load-file-list
 (:require-file-list
   (:lsp "C:/.ru/cad/All Users/dcl/Ru_cad.dcl")
   (:prj "C:/.ru/cad/Source/Lisp/LIB/ru-lib-all.prj")
);_ end of :require-file-list
 (:ob-directory)
 (:tmp-directory)
 (:optimization . st)
);_ end of PRV-DEF

;; EOF
```

В этот проект мы включили один LSP-файл, один DCL-файл, один PRJ-файл (целый проект, в котором есть свой список файлов), задали режим использования ActiveX и отдельное пространство имен для приложения. Во время построения приложения в окне **Build Output** (Сообщения сборки) мы прочитали массу любопытных "интимных" подробностей о наших файлах и получили итоговый файл с расширением vlx.

Считается, что VLX-приложение обеспечивает безопасность и производительность. Безопасность, мол, за счет шифрованного двоичного кода, а производительность за счет более эффективного выполнения откомпилированного кода. Предположим, что это так (в отношении скорости — без всяких "предположим"). А в чем же выгода по сравнению с FAS-приложениями, у них точно такая же безопасность и скорость? И там и тут не настоящий откомпилированный код, псевдокомпилированный, доступный для исполнения только при загруженной среде исполнения Visual LISP.

Достоинством VLX-приложений считается то, что все приложение состоит из одного файла, а при построении приложения *с собственным пространством имен* внутренние функции и переменные не конфликтуют с одноименными символами других приложений. Глобальные переменные такого VLX-приложения будут доступны только внутри загрузившего их приложения. За счет этого могут быть получены интересные эффекты. Внутри каждого VLX-приложения с отдельным пространством имен может быть и своя локальная функция `*error*`, о восстановлении которой не надо заботиться, и может использоваться функция `vl-exit-with-error` с выходом на внешний обработчик, а также функция `vl-exit-with-value`, возвращающая код ошибки без прерывания работы программы.

Для отдельного автономного приложения это справедливо, но для такой большой системы, как наша, совершенно неприемлемо. Большая система как раз и рассчитана на взаимодействие множества функций, размещенных в разных файлах. Функции VLX-приложения с отдельным пространством имен можно сделать доступными в других приложениях, можно и импортировать в пространство имен VLX-приложения посторонние функции. Но дело это не очень надежное. Помимо лишних сложностей с написанием дополнительного кода всегда есть вероятность, что автор забудет экспортировать или импортировать какую-нибудь функцию. А если вероятность есть, то это произойдет непременно.

Включение ресурсов внутрь VLX-приложения для большой системы также не дает преимуществ. Что мы получим от того, что у нас на диске не будет пары десятков DCL-файлов? Ничего, кроме лишних хлопот. То, что текст DCL спрятан от посторонних глаз, ничего не решает. Ну нет там никаких секретов! Может быть, таким образом мы защитим свою интеллектуальную собственность, ведь на конструирование диалогов (диалоговых окон) иногда затрачивается много времени? Упрятывание диалога в VLX-файл является защитой только от "чайника", диалоги оттуда прекрасно извлекаются. Реальное преимущество было бы в случае, если бы диалог загружался вместе с приложением, но этого нет. Диалог, прикомпонованный к программе, просто как бы находится на путях поиска, вот и все.

А теперь рассмотрим возникающие проблемы. Главная из них в том, что DCL-файлы приходится много раз редактировать, поправлять, просто "наводить блеск" или исправлять ошибки. Иногда это приходится делать на компьютерах пользователей. Один DCL-файл может использоваться в разных программах. Как работать с диалогами, если они будут спрятаны внутри VLX-файлов? Каждый раз перекомпилировать проекты? Может быть, кому-то это и нравится, но в системе ruCAD этого делать мы не будем.

Если сравнить FAS-приложение и VLX-приложение, скомпонованное в режиме **Simple** (Простое), т. е. включающее только LSP-файлы, то обнаружится, что в таком VLX имеется только дополнительный заголовок и несколько дополнительных байт в конце файла. Других отличий нет. Зато FAS-приложение ведет себя полностью как LSP-файл, только достаточно надежно (по крайней мере не так, как "AutoCAD PROTECTED LISP file") защищенный. LSP-файлы (мы об этом уже упоминали) теперь при загрузке также автоматически компилируются.

Решение об использовании FAS-приложений, простых VLX-приложений, сложных VLX-приложений или VLX-приложений с отдельным пространством имен должно приниматься с учетом сложности и области применения. Если автор разрабатывает отдельную программу, которая будет работать неизвестно где и в каком окружении, то оптимальным может оказаться сборка VLX-приложения с отдельным пространством имен. Но мы разрабатываем очень большую систему, в которой:

- около 500 конечных программ;
- почти у всех программ головная функция вызывается с параметрами;
- около 5% программ используют диалоговые окна из DCL-файлов;
- все DCL-файлы используют включаемые DCL-файлы с типовыми группами полей;
- почти все конечные программы используют функции из резидентных библиотек (около 800 функций);

- библиотечные функции (около 5%) также используют DCL-файлы;
- в программах и функциях осуществляются вызовы COM-серверов;
- некоторые программы подгружают специальные дополнительные LISP-библиотеки;
- многие программы используют одни и те же внешние данные;
- все программы используют небольшой набор общих данных (о пользователе, проекте) и многие программы применяют данные из словарей рисунка.
- используются дополнительные WIN-приложения (просто EXE-файлы и COM-серверы в DLL-библиотеках);
- все это необходимо реализовать в кратчайшие сроки.

У нас уже продумана система предотвращения конфликтов имен и определен необходимый, но достаточный набор глобальных переменных, предназначенный для обмена данными между приложениями. Попытка использовать VLX-приложения (даже простые) приведет к тому, что нам потребуется создавать еще несколько сотен проектов и "управляться" с ними. Да на это у нас просто не будет времени!

На основании изложенного мы принимаем решение в своей системе использовать только FAS-приложения и для библиотек функций, и для программ.

Резюме

В данной главе мы рассмотрели только некоторые приемы программирования, которые будем использовать. Напоминаем, что эта книга — не учебник, в котором должно быть изложено все, что входит в программу курса, а только описание технологии решения реальной задачи по разработке реальной системы. На иные интересные особенности программирования в Visual LISP мы постараемся обращать внимание читателей в следующих главах.

ГЛАВА 11

План программирования

Нетерпеливый читатель может возмутиться: "Опять планы? Половина книги прочитана, а мы все планируем! А когда же будем программировать?"

Да, конечно, мы можем сразу же начать программирование. Но программирование чего — рисования формата? А может быть, рисования осей зданий? Или стирания объектов в слое? В *части I* книги мы составили план задач, которые нужно решить, теперь мы займемся составлением плана разработки программ и библиотек функций. Иначе, имея всего четыре руки и две головы, мы сорвем всю работу, хватаясь за то, что придет в голову, и занимаясь бесконечными переделками. Если же у вас имеется большой коллектив разработчиков, то планирование тем более необходимо.

В этой маленькой главе мы еще раз передохнем перед "стартом" и подумаем, в какую сторону нам надо "бежать".

Как установить очередность разработки программ и библиотек

Очередность разработки мы будем устанавливать исходя из того, что нам как можно раньше необходимо получать полезные результаты. Желательно, с демонстрацией заказчику или "начальству". Для того чтобы что-то увидеть на экране нам необходимо:

- запускать систему AutoCAD, но не "саму по себе", а с нашим профилем и с загрузкой нашего (пусть эскизного) меню;
- при создании или открытии нового рисунка система должна сразу устанавливать свойства рисунка, о которых мы рассуждали в *главе 2* (важнейшими из свойств являются масштаб чертежа и единицы рисунка).

После решения этих первоочередных задач мы уже сможем что-то рисовать. Решение первых двух задач сразу потребует разработки множества функций.

- *Во-первых*, система ruCAD должна уметь определять расположение любого своего файла. Для этого, как минимум, необходимо уметь читать информацию из реестра и на ее основе оперировать с именами файлов и каталогов. Такие функции должны разрабатываться в первую очередь.

- *Во-вторых*, при загрузке меню нам потребуется установка доступа к его пунктам с учетом прав пользователей. Примерный набор функций для работы с меню мы уже составили в *главе 8*. Это весьма непростые функции и мы должны включить их в первую очередь разработки.
- *В-третьих*, нам сразу потребуются функции ввода данных. Полагаться на стандартные `get`-функции языка LISP мы не будем.
- *В-четвертых*, нам потребуются функции для поддержки задуманной системы пропорциональности — определения размеров внемасштабных символов в зависимости от масштаба чертежа и наименования единиц измерения (которые также надо устанавливать и запоминать).

Все эти группы функций потребуют множества вспомогательных функций, которые придется создавать по ходу работ. Формирование первоочередных функций описано в *главе 13*.

Далее нам понадобятся специальные функции, реализовать которые только на LISP невозможно — для вывода диалоговых окон, написанных с использованием COM-технологий. Откладывать это "на потом" нельзя, т. к. многие из них определяют стратегию разработки нашей системы. Если мы, например, не сможем реализовать работу с иллюстрированными меню, то придется возвращаться к трудоемким для программиста традиционным технологиям, т. е. сорвать сроки разработки.

Разработке таких функций будут посвящены главы 14—21.

После того, как мы обзаведемся требуемым набором базовых функций, можно смело продолжать разработку. Пополнение библиотек описано в *главе 22*.

Какие библиотеки будем создавать в первую очередь

В первую очередь мы будем формировать главную LISP-библиотеку и связанный с ней набор вспомогательных приложений. Работа над этой библиотекой никогда не будет завершена. Даже после выпуска системы в свет пополнение и совершенствование библиотеки будет продолжаться.

Библиотеки функций для прикладных программ заранее планировать бесполезно. По мере разработки программ придется выявлять повторяющиеся участки кода, по возможности преобразовывать в более общий вид и выносить в библиотеки.

Какие программы нам понадобятся в первую очередь

Разработка библиотек не является самоцелью. Конечным результатом являются прикладные программы. Желательно начать их формирование как можно быстрее, но для ускорения процесса нам придется предварительно разработать программы, облегчающие труд самих программистов. О том, как хороший инструмент может радикально изменить труд разработчика, мы уже писали в *главе 10*, когда упоминали программу ruLispExplorer. Программистам или, по их поручению, толковым пользо-

вателям-активистам придется проделать большую работу, не связанную с непосредственным написанием текста программ. Потребуется и редактирование меню, и создание типов линий, и подготовка блоков и библиотек блоков, и разработка штриховок, и конвертирование файлов в новые форматы. К автоматизации таких процессов надо приступить в первую очередь. Впоследствии эти программы пригодятся и конечным пользователям.

После того, как мы надежно обеспечим собственную работу, можно браться за прикладные программы. Теперь критерием очередности должна быть максимальная эффективность конечного результата при минимуме трудозатрат на разработку.

Рекомендуем еще раз просмотреть *главу 2*, в которой мы занимались постановкой задачи, составляли реестр задач, выявляли приоритеты, превращали специальные задачи в общие. Если мы отнеслись к этому формально на этапе постановки, то теперь настанет час (вернее, мучительные месяцы) расплаты.

Не нужно сразу браться за разработку задач по рисованию сложных изображений по конкретному разделу проекта. На любой такой программе можно задержаться на много дней, а то и месяцев.

Примечание

Вспоминается, как в одном очень сильном проектном институте во времена первоначального внедрения AutoCAD взялись автоматизировать вычерчивание чертежей закладных деталей в железобетонных колоннах. Строители знают, сколько времени уходит на эти чертежи, но знают и степень актуальности задачи. Пока все силы небольшого коллектива программистов были брошены на "закладушки", успел распасться СССР, прекратилось строительство, требующее таких колонн, сами программисты разбежались по банкам, а оставшиеся проектировщики так и работают в "голой" системе AutoCAD.

В первую очередь мы будем пополнять меню простенькими мини-программами (о них мы упоминали в *главе 10*). Разработка в этом случае будет сводиться к созданию простейшей программы, наподобие приведенной в листинге 10.1, и формированию XML-меню для вызова такой программы со множеством разнообразных параметров. Эту монотонную работу можно кому-то поручить. Обычно такие программы основаны на вставке блоков. Внедрение таких программ поможет надолго занять нетерпеливых пользователей, которых нужно научить правильно готовить библиотеки блоков и самостоятельно подключать к системе через иллюстрированные меню.

Следующим "слоем" программ должны стать приложения для облегчения оформления общих данных по рабочим чертежам. Общие данные оформляют все, работа это достаточно простая, но "муторная". Хотя для различных разделов проекта существует своя специфика, но с точки зрения программиста все это однотипные задачи по заполнению текстами таблиц, вставке указаний и примечаний. Единые базовые программы позволят пользователям самим расширять нашу систему за счет подготовки типовых таблиц и текстов.

Далее необходимо обеспечить пользователей средствами рисования простых изображений — текстов, выносок, обозначений позиций. Множество разнообразных задач решаются с использованием небольшого набора единых базовых функций, например, рисования различных трасс.

Важнейшей задачей, значительно ускоряющей проектирование, является составление спецификаций оборудования. Этому направлению мы посвятим *главу 32*. При

программировании работы со спецификациями важно не допустить распространенных ошибок — разработки специальных программ для каждой специальности и попыток автоматического определения объемов работ.

После перехода к разработке специализированных программ также важно не поддаться на уговоры, угрозы или шантаж заинтересованных специалистов и не начать длительную разработку не самых актуальных задач. Разработка программ для рисования любых трубопроводов "в одну линию" важнее, чем для рисования "в три линии", т. к. облегчит труд большего числа специалистов. В свою очередь, деталировочное рисование трубопроводов должно быть автоматизировано раньше, чем автоматизация вычерчивания установок центральных кондиционеров.

Как выполнять тестирование

Мы не случайно остановились на очередности разработки программ. Наша система не должна "вдруг" появиться в полностью "готовом к употреблению" виде. Таким образом она может появиться только для совершенно посторонних организаций, а во время разработки все компоненты должны как можно раньше начинать использоваться у заказчика. Только реальная работа в реальных производственных условиях позволяет судить о качестве программы.

Программа, как и любое изделие, должна проходить различные виды испытаний. Первые испытания функции проводит автор. Здесь важно не поддаться искушению доказать работоспособность функции, а попытаться выявить ее непригодность. Достигается это использованием всех допустимых вариантов параметров. Примеры вызова функций в различных вариантах лучше всего вписывать в комментариях, так, чтобы в Visual LISP можно было выделить пример и загрузить в систему AutoCAD.

Второму этапу испытаний функция будет подвергаться при тестировании других функций, в которых она используется. На этом этапе часто выявляется неверный подход при первоначальном испытании. Например, функция преобразования строки в логический тип первоначально выглядела так:

```
(defun ru-conv-str-to-bool (s)
;; Пример: (ru-conv-str-to-bool "0")
 (member (strcase s) '("1" "Y" "" "YES" "ДА" "Д" "T" "TRUE" ".T."))
);_ end of defun
```

Казалось, что все правильно. Компактно и просто. Проверено, является ли строка членом списка допустимых строк. В результате функция должна вернуть `T` или `nil`. Конечно, мы проверяли эту функцию и знали, что функция `member` возвращает часть списка, начинающуюся с первого найденного выражения. Если, например, выполнить

`(ru-conv-str-to-bool "1")`, то будет возвращено

`("1" "Y" "YES" "ДА" "Д" "T" "TRUE" ".T.")`

Это не `nil`, но и не `T`, а список. Тестирующий пример не способствовал фиксации внимания на такой детали. Последствия этого нюанса выявились только при тестировании другой функции, в которой нужно было получить именно логический тип для передачи COM-серверу при чтении переменной из INI-файла. В INI-файле вся

информация хранится в виде строк. "Логические строки" обычно записываются в виде "1" (истина) или "0" (ложь). Обычно, но не всегда. Некоторые программы могут написать что-то другое, для этого и был введен список возможных "строковых истин". Как только понадобилось в иную программу передать "истинную истину", а не "не ложь", так и обнаружилась ошибка. Функцию преобразования пришлось переписать следующим образом:

```
(defun ru-conv-str-to-bool (s)
;; (ru-conv-str-to-bool "0") вернет nil
;; (ru-conv-str-to-bool "1") вернет T
 (not (null (member (strcase s)
              '("1" "Y" "" "YES" "ДА" "Д" "T" "TRUE" ".T.")
            );_ end of member
     )
)
)
```

Этот пример показывает, что алгоритмические ошибки возможны даже в столь простых ситуациях, а выловлены они могут быть с большим трудом.

Третий этап испытаний функции будет проходить в работающих программах. Ненадежность авторских испытаний мы уже показали. Готовая программа является более сложной структурой и лучше всего ее испытывать в "боевых" условиях. Для этого нужно создать группу пользователей-испытателей. Пусть они просто выполняют свою реальную работу, используя постоянно пополняющиеся компоненты нашей системы. Работа должна быть именно реальной, а не надуманной, потому что по неведомым человеку законам в испытательных лабораториях все работает "как нужно", а в реальных условиях — "как всегда".

В группе добровольцев должны быть и "обезьяна с гранатой", и "тетки", и "чайники", и "ламеры". Все они, в меру способностей и зловредности, могут попытаться нарушить работу программ или работать правильно. От них требуется не доказательство работы, а выявление сбоев и помощь в поиске сбойных ситуаций. Для этого понадобится выполнение отладочных мероприятий, о которых мы писали в *главе 10*.

Этап реальных испытаний быстро избавляет программистов от излишней самоуверенности и необоснованного оптимизма в программных решениях.

ГЛАВА 12

Формирование каркаса ruCAD

После окончательного уточнения плана работы мы можем приступать к разработке. Но предварительно необходимо сформировать каркас будущей системы — создать систему каталогов, подготовить вспомогательные файлы, создать временные файлы для запуска системы.

Создание системы каталогов

Планируемую систему каталогов мы обсуждали *в главе 2*. Напомним, что на период разработки вся система будет базироваться в одном каталоге c:\.ru\cad и его подкаталогах у всех разработчиков. Разумеется, все они должны иметь права администратора на своих компьютерах. Где-то на сервере должна находиться текущая последняя версия системы, ежедневные архивы обновлений от каждого разработчика. Ежедневно должна производиться синхронизация версий. Организацию этой работы надо тщательно продумать, чтобы предотвратить "затирание" разработок разных авторов.

Для того чтобы при архивировании и синхронизации не терялись пустые каталоги, необходимо в каждый из них поместить файлы dirinfo.ini с комментариями к файлам и каталогам. По мере разработки система каталогов расширяется в глубину. Перечень каталогов, примерно в середине разработки, приведен в листинге 12.1. Полужирным шрифтом выделены каталоги, путь к которым записывается в реестр. При установке системы у конечных пользователей эти каталоги могут оказаться в разных местах. После имени каталога в кавычках указано, как именуется этот каталог в реестре, в круглых скобках — комментарий к каталогу, а в квадратных скобках — пример возможного полного имени каталога при установке для пользователя с именем "UserName" в операционных системах Windows 2000 или Windows XP. Более подробные комментарии к именам каталогов приведены в *главе 3*.

Листинг 12.1. Перечень каталогов системы

```
c:\.ru\cad —"RootDir" (корневой каталог системы)
[c:\Program Files\ru\CAD]
%RootDir%\bin (двоичные файлы)
```

c:\.ru\cad\All Users — "AllUsersDir" (данные для всех пользователей)
[c:\Documents and Settings\All Users\ruCAD]
%AllUsersDir%\app (FAS-файлы приложений и библиотек)
%AllUsersDir%\block (блоки в отдельных файлах)
%AllUsersDir%\block-lib (библиотеки блоков)
%AllUsersDir%\block-lw (блоки с изменяемой шириной линий)
%AllUsersDir%\dcl (диалоговые окна)
%AllUsersDir%\help (справочные файлы)
%AllUsersDir%\table (блоки и описания таблиц)
%AllUsersDir%\Temp (временные файлы)
%AllUsersDir%\template (шаблоны различных файлов)
%AllUsersDir%\txt\Типовые тексты (типовые тексты для разных разделов)

c:\.ru\cad\All Users\html — "HtmlDir" (HTML-файлы)
[c:\Documents and Settings\All Users\ruCAD\html]

c:\.ru\cad\All Users\Layers\Все — "LayersClassDir" (классификатор слоев)
[c:\Documents and Settings\All Users\ruCAD\Layers\Все]

c:\.ru\cad\All Users\xml\images — "XmlImagesDir" (иллюстрации к меню)
c:\.ru\cad\All Users\xml\menu — "XmlMenuDir" (XML-меню)
%XmlMenuDir%\options (меню опций для программ)
%XmlMenuDir%\snip (меню нормативных и ссылочных документов)
%XmlMenuDir%\txt (меню типовых текстов)

c:\.ru\cad\Application Data — "AppDataDir" (данные посторонних приложений)
[c:\Documents and Settings\Application Data]

c:\.ru\cad\Local Settings —"LocalSettingsDir" (данные приложений для текущего пользователя)
[c:\Documents and Settings\UserName\ruCAD]

c:\.ru\cad\Local Settings\Application Data\ruCAD — "LocalAppDataDir" (данные приложений ruCAD для текущего пользователя)
[c:\Documents and Settings\UserName\ Application Data\ruCAD]
%LocalAppDataDir%\CoordEditor (настройки редактора координат)
%LocalAppDataDir%\DataLinks (файлы подключений к источникам данных)
%LocalAppDataDir%\FileDialog (настройки файлового диалога)
%LocalAppDataDir%\Format (настройки мастера рисования форматов)
%LocalAppDataDir%\Format\Blocks (блоки основных надписей)
%LocalAppDataDir%\Format\Logo (логотипы проектных организаций)
%LocalAppDataDir%\GetString (настройки редакторов строк)
%LocalAppDataDir%\ini (настроечные файлы)
%LocalAppDataDir%\LayerClassExplorer (настройки классификатора слоев)
%LocalAppDataDir%\LispExplorer (настройки навигатора функций)
%LocalAppDataDir%\ListViews (настройки редактора списков)
%LocalAppDataDir%\Starter (настройки программы-стартера)
%LocalAppDataDir%\TipsOfDay (настройки "Советов дня")
%LocalAppDataDir%\XMLEditor (настройки редактора XML)

c:\.ru\cad\Local Settings\Application Data\ruCAD\AutoCAD — "LocalAcadAllVersionDir" (данные AutoCAD для текущего пользователя, общие для всех версий)

[c:\Documents and Settings\UserName\ Application Data\ruCAD\AutoCAD]

%LocalAcadAllVersionDir%\15

%LocalAcadAllVersionDir%\16

c:\.ru\cad\Local Settings\Current User — "CurrentUserDir"

[c:\Documents and Settings\UserName\ruCAD]

%CurrentUserDir%\COMMON (настройки для общей рабочей группы)

%CurrentUserDir%\Guest (настройки для группы "гостей")

%CurrentUserDir%\TOPO (настройки для группы топографов)

%CurrentUserDir%\Water (настройки для группы АС)

c:\.ru\cad\samples (примеры использования системы — могут располагаться рядом с рабочими файлами)

[c:\Documents And Settings\UserName\Мои документы\ruCAD\Samples]

c:\.ru\cad\Source\Lsp — "LspSourceDir" (исходные тексты, которые могут быть только у разработчиков, расположение любое)

Как видим, у конечного пользователя система может быть рассредоточена по множеству каталогов. Конечно, это не очень удобно, но возврата в "добрые старые времена", когда любые файлы можно было располагать где угодно, уже не будет.

Читателям, терзаемым сомнениями, мы рекомендуем еще раз перечитать *главу 3*.

Запись в реестр Windows

Для того чтобы наша система знала, "где что лежит", в реестр Windows должны быть занесены указания о расположении компонентов. Запись следует производить в раздел HKEY_CURRENT_USER\SOFTWARE (именно сюда, а не в другие "подходящие" места). Запись в реестр должна будет сделать программа установки системы. Во время разработки записи в реестр мы будем осуществлять вручную, а для облегчения процесса создадим командный файл (листинг 12.2), который будем периодически дополнять. Если сравнить листинг 12.2 с листингом 3.3, то можно заметить изменения, внесенные в процессе разработки.

Листинг 12.2. Файл ruCAD.reg

```
REGEDIT4

[HKEY_CURRENT_USER\SOFTWARE\ruCAD group\ruCAD]
"RootDir"="C:\\.ru\\CAD"
"AppDataDir"="C:\\.ru\\CAD\\Application Data"
"LocalSettingsDir"="C:\\.ru\\CAD\\Local Settings"
"LocalAppDataDir"="C:\\.ru\\CAD\\Local Settings\\Application Data\\ruCAD"
"LocalAcadAllVersionDir"="C:\\.ru\\CAD\\Local Settings\\Application
Data\\ruCAD\\AutoCAD"
"AllUsersDir"="C:\\.ru\\CAD\\All Users"
```

```
"CurrentUserDir"="C:\\.ru\\CAD\\Local Settings\\Current User"
"LspSourceDir"="C:\\.ru\\CAD\\source\\lisp"
"HtmlDir"="C:\\.ru\\CAD\\All Users\\Html"
"XmlMenuDir"="C:\\.ru\\CAD\\All Users\\Xml\\Menu"
"XmlImagesDir"="C:\\.ru\\CAD\\All Users\\Xml\\Images"
"ruCADVersion"="ruCAD LT  version for book"
"TxtEditor"="c:\\Program Files\\Aditor\\aditor.exe"
"LayersClassDir"="c:\\.ru\\cad\\All Users\\Layers\\Все"
"AcadAppString"="AutoCAD.Application.15"
"UserConnectionString"="Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\\.ru\\cad\\Local Settings\\Application Data\\ruCAD\\ru_users.mdb;Persist
Security Info=False"
```

В приведенных примерах читателям пока непонятно назначение многих папок и ключей реестра. Все разъяснится по мере прочтения следующих глав.

Создание временного ярлыка

В *главе 3* мы наметили запуск системы из специальной программы-стартера. Пока стартер нами не изготовлен, придется запускать AutoCAD с помощью обычных ярлыков, располагаемых на рабочем столе или панели задач. Проще всего это сделать, скопировав ярлык системы AutoCAD, переименовав его в "ruCAD" и изменив параметры запуска. Главное, что нам нужно, — это задать наш профиль с именем `ruCAD`. Для этого необходимо вызвать диалоговое окно редактирования свойств ярлыка и на вкладке **Ярлык** в текстовом поле **Объект** после полного имени файла acad.exe добавить параметры командной строки:

```
"C:\Acad\2002\acad.exe" /nologo /p "C:\.ru\CAD\ruCAD.arg"
```

Такие параметры задают запуск системы AutoCAD без заставки с профилем, параметры которого читаются из ARG-файла. В ARG-файле, который можно получить экспортом из реестра или из AutoCAD, записаны все настройки профиля. Все настройки нам не особенно интересны, главное, чтобы в нашем профиле "наши" каталоги с файлами поддержки находились раньше остальных каталогов. Перечень каталогов, по которым производится поиск файлов поддержки в ключе `ACAD`, приведен в листинге 12.3. Для улучшения восприятия единая строка перечня разбита на подстроки. Подкаталог `AutoCAD\\15` (для версии 2004 — `AutoCAD\\16`) должен обязательно быть записан перед подкаталогом `AutoCAD`!

Листинг 12.3. Порядок перечисления каталогов

```
"ACAD"=
"C:\\.ru\\cad\\Local Settings\\Application Data\\ruCAD\\AutoCAD\\15;
C:\\.ru\\cad\\Local Settings\\Application Data\\ruCAD\\AutoCAD;
C:\\ACAD\\2002\\support;
C:\\ACAD\\2002\\fonts;
C:\\ACAD\\2002\\help;
C:\\ACAD\\2002\\express;"
```

Пробный запуск системы и ручная настройка профиля ruCAD

Практически необходимо сначала запустить систему AutoCAD, создать профиль `ruCAD`, добавить в окне настройки нужные каталоги в **Support File Search Path** (Пути доступа к файлам поддержки) и экспортировать профиль в ARG-файл.

Создание временного файла acaddoc.lsp

О назначении и содержании предварительного варианта файла acaddoc.lsp мы писали в *главе 10* (см. листинг 10.12). Там мы говорили, что будем создавать этот файл, а теперь пришла пора написать его и поместить в каталог

```
c:\.ru\cad\Local Settings\Application Data\ruCAD\AutoCAD
```

Разработка временного меню для тестирования системы

В *главе 8* мы много рассуждали о том, каким должно быть меню системы ruCAD. Настала пора выполнять обещания. Редактировать файл меню нам придется сотни раз. Не надо забывать, что мы разрабатываем систему сразу для двух версий AutoCAD. Откомпилированные файлы меню версий R15 (2002) и R16 (2004) не совместимы между собой, поэтому мы и предусмотрели отдельные подкаталоги для каждой версии. Но исходный MNU-файл может быть единым. Мы предусмотрим блокировку некоторых специфичных для AutoCAD 2004 команд, чтобы они были недоступны в AutoCAD 2002.

Исходный файл ruCAD.mnu мы поместим в каталог %AllUsersDir%\template и редактировать будем только этот файл. Специальная программа перегрузки меню будет удалять все производные файлы (ruCAD.mns, ruCAD.mnr, ruCAD.mnc) из каталога рабочей версии AutoCAD, копировать в этот каталог ruCAD.mnu и загружать, предотвращая лишние вопросы.

В качестве прототипа файла ruCAD.mnu мы возьмем меню предыдущей системы BestIA, с помощью программы замены строк, упоминавшейся в *главе 9*, изменим имена вызываемых функций, а дальнейшую "доводку" меню будем выполнять вручную по мере разработки программ.

Установка библиотеки DOSLib

В своей системе, по крайней мере в первое время, мы будем использовать функции библиотеки DOSLib. Об установке этой библиотеки придется позаботиться сразу. Применяться она может не только в нашей системе, поэтому DOSLib необходимо установить в обычном порядке в каталоги поддержки всех используемых версий AutoCAD.

Командный файл для регистрации COM-серверов

В последующих главах мы будем разрабатывать множество COM-серверов в виде так называемых ActiveX DLL. Каждая такая библиотека требует регистрации на компьютере. Регистрация библиотек в основном будет производиться из среды Borland Delphi, но для того, чтобы не забыть о какой-то библиотеке, мы будем заносить команды регистрации в специальный командный файл ruCADRegSvr.bat (листинг 12.4). Размещаться этот файл должен в подкаталоге bin рядом с самими DLL-файлами. Лишнее его выполнение не принесет вреда, но если какая-то библиотека окажется незарегистрированной, дефект будет устранен.

Листинг 12.4. Файл ruCADRegSvr.bat

```
regsvr32 AxAcadStatusBarButton.dll
regsvr32 ruADOConnSvr.dll
regsvr32 ruAxSvr.dll
regsvr32 ruCheckListSrv.dll
regsvr32 ruCoordEditSvr.dll
regsvr32 ruDoubleListSrv.dll
regsvr32 ruDualListSrv.dll
regsvr32 ruEditUsersSvr.dll
regsvr32 ruFileInfoSvr.dll
regsvr32 ruFileSearchSvr.dll
regsvr32 ruFolderDlgSvr.dll
regsvr32 ruFormatWizardSrv.dll
regsvr32 ruIniRegSrv.dll
regsvr32 ruShellFileDlgSvr.dll
regsvr32 ruSingleListSrv.dll
regsvr32 ruSplashSvr.dll
regsvr32 ruSysInfoSvr.dll
regsvr32 ruTipsSrv.dll
regsvr32 ruTreeDirSrv.dll
regsvr32 ruTxtViewSrv.dll
regsvr32 ruUserLoginSvr.dll
regsvr32 ruXmlMenuSrv.dll
regsvr32 ruXmlTableSrv.dll
```

Читатели пока могут не обращать внимания на незнакомые имена файлов — со всеми ими мы познакомимся в *главе 15*.

Файл ruCADRegSvr.bat будет заодно являться шпаргалкой при разработке сценария инсталляции системы.

ГЛАВА 13

Разработка первоочередных библиотечных функций

Приступаем к разработке библиотечных функций в соответствии с намеченным в *главе 11* "генеральным" планом. В этой главе мы будем стараться давать самые минимальные пояснения, но некоторые сложные решения все-таки разберем достаточно подробно.

Не все исходные тексты мы сможем разместить в книге, постараемся привести только наиболее важные, показывающие, как реализуется концепция системы. Исходные тексты программ читатели найдут на прилагаемом компакт-диске. Иногда в исходных текстах применяются функции, еще не описанные в книге — в подобных случаях, при реальной разработке мы иногда включаем такие неразработанные функции в виде программных "заглушек", возвращающих константы.

Авторами большинства функций являются члены объединения ruCAD Group — Сергей Зуев, Петр Лоскутов, Николай Полещук, Александр Якушин. При использовании решений других авторов будут приводиться дополнительные комментарии. Некоторые исходные тексты могут показаться читателям знакомыми, и это не должно смущать, т. к. множество решений в языке LISP являются типовыми и очевидными, например преобразование углов. А ведь было время, когда и эти простейшие функции приходилось изобретать самостоятельно!

Надеемся, что читатели еще не забыли про соглашение об именовании функций и переменных *(см. главу 9)* и не будут возмущаться примененным в некоторых местах неудобным форматированием текстов — это вызвано особенностями издания и стремлением опубликовать побольше примеров в исходных текстах.

Замечание

По ходу разработки реализация многих функций неоднократно будет меняться, в том числе и в самый последний момент. Отслеживать историю изменений мы не имеем возможности, поэтому не удивляйтесь, если на компакт-диске окажутся совсем другие тексты.

Расположение компонентов системы

Наша система, в зависимости от версии Windows и пользовательских настроек, может размещаться в нескольких каталогах. Имена каталогов записываются в систем-

ный реестр во время инсталляции системы. Для определения расположения любого файла и потребуются рассматриваемые функции.

Все функции, определяющие расположение компонентов, очень просты и, как правило, выполняют *конкатенацию*[1] имен основных каталогов, описанных в реестре, и подкаталогов или имен файлов. Определение основных каталогов, по мере реализации системы, выполняется разными методами. На первоначальном этапе мы будем использовать прямое чтение из реестра, в последующем переопределим некоторые функции для использования единого COM-сервера, а в самом конце придем к анализу структурированного списка, передаваемого программой-стартером.

В качестве примера рассмотрим вариант чтения реестра.

Для *инкапсуляции*[2] в одном месте имени ключа реестра с параметрами системы ruCAD напишем простенькую функцию, возвращающую строковую константу (листинг 13.1).

Листинг 13.1. Функция `_ru-dirs-hkcu-reg-key`

```
(defun _ru-dirs-hkcu-reg-key ()
;|-----------------------------------------------------------------------
Возвращает имя ключа реестра с переменными нашей системы
-----------------------------------------------------------------------|;
  "HKEY_CURRENT_USER\\SOFTWARE\\ruCAD group\\ruCAD"
)
```

Корневой каталог системы мы пока будем определять так, как показано в листинге 13.2.

Листинг 13.2. Функция `ru-dirs-get-root`

```
(defun ru-dirs-get-root (/ result reg_key)
;|-----------------------------------------------------------------------
Возвращает корневой каталог ruCAD
-----------------------------------------------------------------------|;
(setq reg_key (_ru-dirs-hkcu-reg-key))
(if (setq result (vl-registry-read reg_key "RootDir"))
;; избавляемся от временной глобальной переменной
  (if *ru_root_dir* (setq *ru_root_dir* nil))
   (progn
;; На крайний случай, если в реестре нет информации, сообщаем
    (ru-msg-alert (strcat "Не записан в реестр RootDir в \n" reg_key))
;; и записываем. А берем из временной глобальной *ru_root_dir*
    (if *ru_root_dir*
      (setq result (vl-registry-write reg_key "RootDir" *ru_root_dir*)
           *ru_root_dir* nil)
```

[1] Конкатенация — сцепление, соединение.

[2] Инкапсуляция — механизм, объединяющий данные и код, манипулирующий этими данными. В контексте данной книги мы скрываем в функциях методы получения требуемых данных (константа или вычисления). Это, конечно, не то, что понимается под концепцией инкапсуляции в объектно-ориентированном программировании.

```
      (setq result (vl-registry-write reg_key "RootDir"
          "C:\\Program Files\\.ru\\CAD"))
    );_ if
 );_ progn
 );_ if
 (ru-dirs-path-and-slash result)
);_ defun
```

Здесь мы использовали функцию `ru-dirs-path-and-slash`, добавляющую слэш к имени каталога (листинг 13.3). Эта операция выполняется часто, поэтому выносим ее в отдельную функцию.

Листинг 13.3. Функция `ru-dirs-path-and-slash`

```
(defun ru-dirs-path-and-slash (dir)
;|-----------------------------------------------------------------------
 Возвращает имя каталога со слэшем на конце
-----------------------------------------------------------------------|;
(strcat (vl-string-right-trim "\\" dir) "\\")
);_ defun
```

По аналогии с функцией `ru-dirs-get-root` написаны и другие функции, листинги которых мы не приводим:

- `ru-dirs-get-all-users` — возвращает каталог данных для всех пользователей;
- `ru-dirs-get-app-data` — возвращает каталог данных приложений;
- `ru-dirs-get-current-user` — возвращает каталог данных текущего пользователя;
- `ru-dirs-get-html` — возвращает каталог HTML-файлов;
- `ru-dirs-get-layers-class` — возвращает каталог классификатора слоев;
- `ru-dirs-get-local-acad-all-version` — возвращает каталог файлов поддержки AutoCAD;
- `ru-dirs-get-local-app-data` — возвращает каталог локальных данных приложений;
- `ru-dirs-get-local-settings` — возвращает каталог локальных настроек;
- `ru-dirs-get-lsp-source` — возвращает каталог исходных текстов.

После определения каталогов, которые могут "жить сами по себе", мы можем определять имена подкаталогов с обязательными именами, например так, как показано в листинге 13.4. Во всех функциях, возвращающих имена каталогов, действует соглашение — имя каталога всегда возвращается с замыкающим слэшем.

Листинг 13.4. Функция `ru-dirs-menu-xml`

```
(defun ru-dirs-menu-xml ()
;|-----------------------------------------------------------------------
Возвращает подкаталог XML
-----------------------------------------------------------------------|;
 (strcat (ru-dirs-get-all-users) "xml\\")
);_defun
```

Далее мы можем написать функции для определения полных имен файлов. Все они очень простые. Приведем несколько примеров таких функций (листинги 13.5—13.8).

Листинг 13.5. Функция `ru-file-acad`

```
(defun ru-file-acad (name all_version)
;|-------------------------------------------------------------------------
Возвращает полное имя файла name в каталогах файлов поддержки AutoCAD нашей
системы. Если параметр all_version не nil, то возвращается имя файла в подкаталоге
запущенной версии AutoCAD (15 или 16)
-------------------------------------------------------------------------|;
(strcat (ru-dirs-acad all_version) name)
);_ end of defun
```

Листинг 13.6. Функция `ru-file-dwgname`

```
(defun ru-file-dwgname ()
;|-------------------------------------------------------------------------
Возвращает полное имя текущего рисунка без расширения
-------------------------------------------------------------------------|;
(strcat (getvar "DWGPREFIX") (vl-filename-base (getvar "DWGNAME")))
);_ end of defun
```

Листинг 13.7. Функция `ru-file-dwgname-type`

```
(defun ru-file-dwgname-type (ext /)
;|-------------------------------------------------------------------------
Возвращает имя рисунка с расширением ext
-------------------------------------------------------------------------|;
(ru-file-set-ext (ru-file-dwgname) ext)
);_ end of defun
```

Листинг 13.8. Функция `ru-file-dwg-not`

```
(defun ru-file-dwg-not ()
;|-------------------------------------------------------------------------
Возвращает имя рисунка с расширением not
-------------------------------------------------------------------------|;
(ru-file-dwgname-type ".not")
);_ end of defun
```

Замечание

К наработке такого большого количества микрофункций можно относиться по-разному, но мы руководствуемся собственным (в том числе и горьким) опытом. Если бы мы разрабатывали *пять* программ, то в каждой из них могли бы написать нечто наподобие `(strcat (getvar "DWGPREFIX") (vl-filename-base (getvar "DWGNAME")))`, и программы были бы понятными. Но когда мы пишем *пять сотен* программ, то тут начинают работать иные законы. Изменение работы или появление новых стандартных функций потребовало бы внесения изменений во множество программ, а нам это давно надоело. Проводя

политику инкапсуляции решений в единственную точку, мы добиваемся гибкости и надежности — ошибка возможна в одном месте, а не в пятистах.

Очень часто используется чуть более сложная функция установки заданного расширения для имени файла (листинг 13.9).

Листинг 13.9. Функция `ru-file-set-ext`

```
(defun ru-file-set-ext (file_name ext)
;|-------------------------------------------------------------------------
 Установка расширения для имени файла
Аргументы:
file_name - имя файла
ext - расширение с точкой или без точки (может быть пустое)
Примеры:
(ru-file-set-ext "c:\\.ru\\cad\\users\\USERS.INI" ".EXE")
Вернет "c:\\.ru\\cad\\users\\USERS.EXE"
(ru-file-set-ext "c:\\.ru\\cad\\users\\USERS.INI" "EXE")
Вернет "c:\\.ru\\cad\\users\\USERS.EXE"
 (ru-file-set-ext "c:\\.ru\\ru_users.mdb" "")
Вернет "c:\\.ru\\ ru_users"
-------------------------------------------------------------------------|;
 (vl-string-trim "\\"
   (strcat (vl-string-right-trim "\\" (vl-filename-directory file_name))
    "\\" (vl-filename-base file_name)
    (if (and ext (/= ext "")) (strcat "." (vl-string-trim "." ext)) "")
   );_ end of strcat
);_ end of vl-string-trim
)
```

Установка свойств рисунка

В списке первоочередных задач находится инициализация системы. При запуске системы и загрузке рисунка многие настройки зависят от его свойств. Прежде всего, это система поддержки пропорциональности символов, базирующаяся на масштабе чертежа и единицах измерения рисунка *(см. главу 3)*. В зависимости от этих параметров производятся все настройки при инициализации системы. При обсуждении стандартов ruCAD мы договорились свойства рисунка хранить внутри файла так, чтобы они были доступны с помощью диалогового окна **Drawing Properties** (Свойства рисунка), вызываемого командой DWGPROPS (СВОЙСТВАРИС). Это диалоговое окно вызывается и при просмотре свойств DWG-файла средствами операционной системы. Напоминаем, что мы договорились отображать и устанавливать дополнительные свойства рисунка на вкладке **Custom** (Прочие):

- Имя свойства "Масштаб 1:" (код 300) — масштаб чертежа, например: `25`.
- Имя свойства "Ед. изм" (код 301) — единица измерения рисунка, например: `мм`.

Полагаться на то, что пользователь правильно введет эти свойства, нельзя (это было бы наивно), поэтому мы предусмотрим более надежные места для хранения базовых свойств, а диалоговое окно будем использовать для ввода и отображения.

В действительности свойства хранятся в специальном словаре, следовательно, нам необходимо:

- научиться читать и записывать свойства программным путем, без диалогового окна;
- создать командный реактор, который после интерактивного ввода данных в диалоговом окне свойств рисунка проверял бы корректность введенных значений.

Необходимо также иметь доступ к свойствам других файлов — например, для просмотра свойств при выборе типового проектного решения *(см. главу 33)*. Желательно и устанавливать свойства "постороннего" файла, не загружая его в AutoCAD, или, как минимум, готовить такое окружение, которое позволило бы при загрузке файла установить для него требуемые свойства.

Мы предусматриваем временное хранение свойств в INI-файле рисунка. Записать и прочитать такие данные мы можем, не открывая рисунка.

Чтение свойств

Прочитать свойства рисунка, отображаемые в диалоговом окне команды DWGPROPS (СВОЙСТВАРИС) в AutoCAD 2002 (особенности AutoCAD 2004 будут рассмотрены позже) очень просто:

```
(setq dwg_props (dictsearch (namedobjdict) "DWGPROPS"))
```

Если в рисунке найден такой словарь, в переменной `dwg_props` будет находиться ассоциированный список примерно такого вида (файл 1st floor.dwg):

```
((-1 . <Entity name: 40094cd8>) (0 . "XRECORD") (5 . "3A8B") (102 .
"{ACAD_REACTORS") (330 . <Entity name: 40089c60>) (102 . "}")
 (330 . <Entity name: 40089c60>) (100 . "AcDbXrecord") (280 . 1)
 (1 . "DWGPROPS COOKIE") (2 . "") (3 . "") (4 . "") (6 . "") (7 . "")
 (8 . "hewetth") (9 . "") (300 . "=") (301 . "=") (302 . "=") (303 . "=")
 (304 . "=") (305 . "=") (306 . "=") (307 . "=") (308 . "=") (309 . "=")
 (40 . 0.113001) (41 . 2.45121e+006) (42 . 2.45126e+006) (1 . "")
 (90 . 0))
```

Свойства рисунка возвращаются в виде структурированного списка, в котором каждое свойство ассоциировано с определенным кодом. Пользовательские свойства имеют коды от 300 до 309 и сохраняются в виде строки формата `Имя=Значение`. Интерпретация пользовательских свойств возлагается на прикладные программы.

В данном случае словарь свойств имеется, но в нем пусто, за исключением свойства с кодом 8 (`Last saved by`). В новом пустом рисунке словаря может и не быть вообще, тогда выражение `(dictsearch (namedobjdict) "DWGPROPS")` вернет `nil`.

Нам нужно обязательно получить свойства, а если они отсутствуют, создать необходимый, но достаточный *набор свойств*.

Запись свойств

Записать список свойств в словарь, на первый взгляд, просто.

При вызове команды DWGPROPS (СВОЙСТВАРИС) в "голой" системе AutoCAD 2002 словарь свойств создается автоматически, но нам нужно создать словарь программ-

но, и предварительно заполнить нужными данными. При этом желательно не испортить данные, которые, возможно, уже имеются в словаре рисунка, полученного откуда-то. Легко было бы удалить прежний словарь вызовом `(dictremove (namedobjdict) "DWGPROPS")`, а потом заполнить своими данными:

```
(dictadd (namedobjdict) "DWGPROPS" (entmakex (list
 '(0 . "XRECORD") '(100 . "AcDbXrecord") '(1 . "DWGPROPS COOKIE")
 (cons 2 "Сеятель") (cons 4 "Товарищ О. Бендер"))))
```

В этом примере мы заполнили поля **Title** (Название) и **Author** (Автор) на вкладке **Summary** (Документ) диалогового окна **Drawing Properties** (Свойства рисунка), оставив пустыми остальные свойства. Так делать не следует — мы могли испортить прежнюю информацию, а как нужно делать — разберем позже.

Особенности AutoCAD 2004

Работа со словарем свойств возможна и в системе AutoCAD 2004, если словарь свойств каким-то образом оказался в файле этой версии — например, при открытии созданного в ранних версиях файла. Выражение

```
(dictsearch (namedobjdict) "DWGPROPS"))
```

может вернуть список свойств, но в диалоговом окне **Drawing Properties** (Свойства рисунка), вызываемым с помощью команды DWGPROPS (СВОЙСТВАРИС), при этом могут отображаться другие данные, а может вернуть `nil`, но окно будет показывать какие-то свойства.

Дело в том, что в AutoCAD 2004 появилось новое свойство объекта активного документа `SummaryInfo`. Если выполнить выражение

```
(vlax-dump-object (vlax-get-property (ru-obj-get-active-document)
 'SummaryInfo) T)
```

то мы получим результат наподобие:

```
; IAcadSummaryInfo: IAcadSummaryInfo Interface
; Property values:
;   Author = "Гипроавтоагрегат. ОТЭ. Исп Зуев С.А. ГИП Корляков Н.Л."
;   Comments = "Дополнительный водомерный узел корпуса 15 КЗКТ."
;   HyperlinkBase = "11111111111111111110001001_RUCAD"
;   Keywords = "КЗКТ водоснабжение корпус 15"
;   LastSavedBy = "User"
;   RevisionNumber = "238.00.01"
;   Subject = " КЗКТ. Корпус 15. 3 очередь"
;   Title = "Водомерный узел № 2"
; Methods supported:
;   AddCustomInfo (2)
;   GetCustomByIndex (3)
;   GetCustomByKey (2)
;   NumCustomInfo ()
;   RemoveCustomByIndex (1)
;   RemoveCustomByKey (1)
;   SetCustomByIndex (3)
;   SetCustomByKey (2)
```

Любопытные подробности:

- этот результат получен в файле, *первоначально созданном в AutoCAD 2002*, загруженном и сохраненном в системе AutoCAD 2004;
- свойства и этого файла, и исходного файла формата AutoCAD 2002 в контекстном меню Проводника Windows XP свойства отображаются правильно;
- *наша* функция диалогового выбора файлов (мы ее будем разбирать в *главе 17*), оснащенная диалоговым окном просмотра свойств файлов, о котором мы *точно знаем*, как оно работает, показывает правильные свойства DWG-файлов всех версий;
- в диалоговом окне команды DWGPROPS (СВОЙСТВАРИС) в системе AutoCAD 2004 отображаются все свойства, записанные нами программно в словарь в AutoCAD 2002;
- функция извлечения свойств (мы ее разберем далее) для AutoCAD 2004 из *нового активного документа* большинство свойств возвращает в виде пустых строк, хотя в диалоговом окне они отображаются правильно;
- функция извлечения свойств для AutoCAD 2004 из этого же документа *после его закрытия* правильно возвращает свойства;
- при сохранении файла из AutoCAD 2004 в формате AutoCAD 2000 и последующем открытии этого файла в AutoCAD 2002 можно убедиться, что свойства вновь записаны в словарь DWGPROPS (СВОЙСТВАРИС);
- выражение `(dictsearch (namedobjdict) "DWGPROPS")` возвращает все наши свойства, но они не отражают изменений, внесенных в диалоговом окне команды DWGPROPS (СВОЙСТВАРИС).

Объект `SummaryInfo` не описан в справочной системе AutoCAD 2004, но исследование его "дампа" и результатов работы команды ACADINFO позволило выявить набор свойств, методов и функций для работы со свойствами документа. Работать с ними удобно, но обнаружена "подлая" ловушка — методы `SummaryInfo` не отображают изменения, внесенные *при ручном редактировании свойств* до тех пор, пока файл не будет сохранен и закрыт. Впрочем, виноват в этом не объект `SummaryInfo`, а команда DWGPROPS (СВОЙСТВАРИС), в чем мы убедимся чуть позже. Только при открытии файла вновь можно увидеть изменения. Рядовому пользователю это незаметно, т. к. он видит свойства только в диалоговом окне, но для программной обработки это "удар ниже пояса".

Не зря, видимо, фирма Autodesk не документировала объект `SummaryInfo`, включив его в продукт, родившийся на год раньше своего года рождения.

Кроме того, можно сделать некоторые предварительные выводы:

- очевидно, что в AutoCAD 2004 информация в объект `SummaryInfo` передается не из словаря, а, скорее всего, в виде результата функции Windows API `StgOpenStorageEx`, рекомендуемой фирмой Microsoft для стандартизированного хранения свойств документов в так называемом *Windows 2000 Structured Storage*;
- при открытии файла прежней версии AutoCAD 2004 копирует информацию из словаря в новое хранилище, не удаляя сам словарь[1];

[1] Это один из наглядных примеров, как нельзя делать программы — раз уж словарь не удаляется из рисунка, то его постоянная обратная синхронизация является, безусловно, *обязательной*. К сожалению, продукция Autodesk дает примеры такого рода.

- возможно будущее изменение технологии работы со свойствами — не одни мы столкнемся с этими недоработками.

Работа с объектом `SummaryInfo` проще и логичнее, особенно с Custom-свойствами, т. к. можно работать с именами свойств, а не с их кодами. Несомненно, использование стандартного хранилища свойств оправдано, жаль только, что спохватились на три года позже, чем надо было — все эти прелести должны были появиться еще в системе AutoCAD 2000.

В результате расплачиваться приходится разработчикам. Нам нужно одновременно поддерживать и старую, и новую технологию, поэтому придется написать функции, возвращающие одинаковые результаты в двух версиях.

Замечание

Эти особенности системы AutoCAD 2004 были обнаружены не сразу. Основная разработка велась нами в AutoCAD 2002, а когда появился AutoCAD 2004, блок функций для работы со свойствами был уже готов. Конечно, мы проверяли работу и в AutoCAD 2004, но загружали в него рисунки, прежде прошедшие через AutoCAD 2002. Работа со свойствами продолжалась через словарь и не вызывала подозрений до тех пор, пока случайно не было обнаружено небольшое несоответствие между свойствами, отображаемыми в диалоговом окне и возвращаемыми функциями. Хороший пример, показывающий необходимость очень тщательного тестирования в разнообразных условиях!

Как решить проблему версии 2004

Решить возникшую проблему возможно несколькими путями.

Во-первых, мы можем отказаться от идеи изменять требующиеся нам важнейшие свойства (масштаб и наименование единиц рисунка) в одном месте с остальными свойствами — диалоговом окне команды DWGPROPS (СВОЙСТВАРИС). В этом случае нам потребуется собственный диалог установки *наших* обязательных свойств, не понадобится использование командного реактора и мы не будем заставлять пользователей придерживаться *нашего* стандарта, в конце концов наши рисунки могут попасть в другое место, где нет нашей системы, да и мы можем использовать посторонние файлы, в которых зарезервированные нами коды пользовательских свойств заняты для других целей. Укромных мест для хранения наших свойств в рисунке найдется достаточно.

Во-вторых, мы можем продолжить поддержку словаря DWGPROPS и в файлах AutoCAD 2004. При этом мы должны отслеживать изменения свойств и дублировать их в словаре, причем для стандартизации решений следует по-прежнему передавать свойства в виде структурированных списков с такими же кодами, как и в AutoCAD 2002, но извлекать свойства по кодам только в AutoCAD 2002. Недостаток этого решения в том, что объект `SummaryInfo` в AutoCAD 2004 ненадежно извлекает измененные в диалоге свойства. Не беремся утверждать, что это происходит всегда и во всех "билдах" AutoCAD 2004, но мы с этим фактом столкнулись и игнорировать его не можем. В результате решение по второму варианту может оказаться и сложным, и ненадежным.

В-третьих, мы можем воспользоваться функциями получения свойств через объекты ObjectDBX, обращаясь к рисунку как к файлу. По отношению к собственному открытому рисунку это выглядит слишком экзотичным, кроме того, файл должен

быть сохранен на диск, т. е. потребуется его принудительное сохранение после каждого применения команды DWGPROPS (СВОЙСТВАРИС).

В-четвертых, мы можем переопределить команду DWGPROPS (СВОЙСТВАРИС), чтобы добиться правильной работы.

Оставим окончательное решение этого вопроса на будущее, а пока, ради любопытства, разберем, как работать со свойствами в AutoCAD 2002.

Вспомогательные функции

Для того чтобы не разбираться с ненужными нам для обработки свойствами, создадим несколько вспомогательных функций. Функция `ru-dwgprops-names-by-codes-map` (листинг 13.10) будет возвращать ассоциированный список стандартных кодов и соответствующих им придуманных нами имен свойств.

Листинг 13.10. Функция `ru-dwgprops-names-by-codes-map`

```
(defun ru-dwgprops-names-by-codes-map ()
  '((2 . "TITLE")(3 . "SUBJECT") (4 . "AUTHOR") (6 . "COMMENTS")
    (7 . "KEYWORDS") (300 . "ACADDWGCUSTOM1") (301 . "ACADDWGCUSTOM2")
    (302 . "ACADDWGCUSTOM3") (303 . "ACADDWGCUSTOM4")
    (304 . "ACADDWGCUSTOM5") (305 . "ACADDWGCUSTOM6")
    (306 . "ACADDWGCUSTOM7") (307 . "ACADDWGCUSTOM8")
    (308 . "ACADDWGCUSTOM9")(309 . "ACADDWGCUSTOM10")
  )
)
```

Функция `ru-dwgprops-rus-names-map` (листинг 13.11) будет возвращать ассоциированный список "наших" имен свойств и соответствующих им русских пояснений, которые пригодятся нам для использования в визуальных целях.

Листинг 13.11. Функция `ru-dwgprops-rus-names-map`

```
((defun ru-dwgprops-rus-names-map ()
  '(("TITLE" . "Название") ("SUBJECT" . "Тема") ("AUTHOR" . "Автор")
  ("COMMENTS" . "Заметки") ("KEYWORDS" . "Ключевые слова")
  ("ACADDWGCUSTOM1" . "        М 1:") ("ACADDWGCUSTOM2" . "    Ед. рис. ")
  ("ACADDWGCUSTOM3" . "Тчк. вставки") ("ACADDWGCUSTOM4" . "Свойство 4")
  ("ACADDWGCUSTOM5" . "Свойство 5") ("ACADDWGCUSTOM6" . "Свойство 6")
  ("ACADDWGCUSTOM7" . "Свойство 7") ("ACADDWGCUSTOM8" . "Свойство 8")
  ("ACADDWGCUSTOM9" . "Свойство 9") ("ACADDWGCUSTOM10" . "Свойство 10")
  )
)
```

Функция `ru-dwgprops-rus-name` (листинг 13.12) будет возвращать русское пояснение к свойству, заданному именем.

Листинг 13.12. Функция `ru-dwgprops-rus-name`

```
(defun ru-dwgprops-rus-name (name)
;| Пример (ru-dwgprops-rus-name "KEYWORDS") вернет "Ключевые слова"
```

```
|;
  (cond ((ru-ent-dxf-code-data (strcase name)
           (ru-dwgprops-rus-names-map)))
        (T (strcat "?" (strcase name)))
 );_ end of cond
)
```

Функция `ru-dwgprops-name-by-code` (листинг 13.13) будет возвращать имя свойства по заданному коду.

Листинг 13.13. Функция `ru-dwgprops-name-by-code`

```
(defun ru-dwgprops-name-by-code (code)
;| Пример (ru-dwgprops-name-by-code 301) вернет "ACADDWGCUSTOM2"
 |;
  (ru-ent-dxf-code-data code (ru-dwgprops-names-by-codes-map))
)
```

Функция `ru-list-reverse-assoc` (листинг 13.14) возвращает ассоциированный список, у элементов которого переставлены местами код и значение.

Листинг 13.14. Функция `ru-list-reverse-assoc`

```
(defun ru-list-reverse-assoc (lst)
;; Пример: (ru-list-reverse-assoc (ru-dwgprops-names-by-codes-map))
  (mapcar (function (lambda (x) (cons (cdr x) (car x)))) lst)
)
```

Листинг 13.15. Функция `ru-list-make-two-lists-from-assoc`

```
(defun ru-list-make-two-lists-from-assoc (assoc_list)
;| (ru-list-make-two-lists-from-assoc
    (ru-list-make-assoc-from-two-lists (list 1 2 3)
      (list "1-й" "2-й" "3-й"))
  )
 ((1 2 3) ("1-й" "2-й" "3-й"))
|;
  (list (mapcar 'car assoc_list) (mapcar 'cdr assoc_list))
)
```

Сохранение свойств в AutoCAD 2002

Словарь свойств создается с помощью функции `entmakex`, которой нужно передать список свойств. Часть существующих свойств должна сохраниться, часть обязательно заменена, несуществующие свойства добавлены. Перед передачей в функцию список должен выглядеть как бы созданным функцией `list`:

```
(list
  '(0 . "XRECORD")
  '(100 . "AcDbXrecord")
```

```
  '(1 . "DWGPROPS COOKIE")
  '(280 . 1)
  '(2 . "Название ")
  '(3 . "Стройка, объект")
  '(4 . "Автор")
  '(6 . "Дополнительные комментарии")
  '(7 . "ключевые слова")
  '(8 . "Кто сохранил")
  '(9 . "")
  '(300 . "М 1:=500")
  '(301 . "Ед. рис. =м")
  '(302 . "Точка вставки=Левый нижний угол")
  '(303 . "Свойство-3=Значение")
  '(304 . "=")
  '(305 . "=")
  '(306 . "=")
  '(307 . "=")
  '(308 . "Свойство-8=Значение")
  '(309 . "Свойство-9=Значение")
)
```

При подготовке этого списка следует произвести все требуемые замены.

Свойства с кодами `(list 2 3 4 6 7 300 301 302 303 304 305 306 307 308 309)` мы можем изменять, заполнять остальные нам не следует, но функции `entmakex` требуется (не всегда, но и не вредит) условно-постоянная часть из
`(0 . "XRECORD"), (100 . "AcDbXrecord")` и `'(1 . "DWGPROPS COOKIE")`.

Удобно иметь функцию хитрого объединения двух ассоциированных списков, которая должна уметь:

- добавлять элементы (точечные пары) списка `list_2` к списку `list_1`;
- если `list_1` равен `nil`, то возвращать `list_2`;
- объединение элементов производить по кодам;
- если для элемента `list_2` с кодом `code_2` имеется "однокодовый" аналог в `list_1`, то значение элемента списка `list_2` с кодом `code_2`, отличное от `nil`, заменяет соответствующее значение в `list_1`, а при `nil` сохраняется прежнее значение в `list_1`;
- если для элемента `list_2` с кодом `code_2` отсутствует аналог в `list_1`, то значение элемента списка `list_2` с кодом `code_2`, отличное от `nil`, добавляется в `list_1`;
- в объединенном списке должны быть только истинные точечные пары[1].

Результат разработки такой функции приведен в листинге 13.16.

Листинг 13.16. Функция `ru-list-union-two-assoc`

```
(defun ru-list-union-two-assoc (list_1 list_2)
  (setq list_1 (vl-remove-if-not 'ru-list-is-dotted-pair list_1))
```

[1] Не результат, подобный `(cons 302 nil)`, возвращающий `(302)`, а не `(302 . nil)`.

```
  (if (setq list_2 (vl-remove-if-not 'ru-list-is-dotted-pair list_2))
    (foreach x list_2
      (setq list_1 (append (vl-remove (assoc (car x) list_1) list_1)
                           (list x)));_ end of setq
   );_ end of foreach
    list_1
 );_ end of if
);_ end of defun
```

Функция `ru-list-is-dotted-pair` (листинг 13.17) проверяет, является ли список точечной парой.

Листинг 13.17. Функция `ru-list-is-dotted-pair`

```
(defun ru-list-is-dotted-pair (lst / x)
  (and (not (atom lst)) (not (listp (cdr lst))))
);_ end of defun
```

Основная функция для AutoCAD 2002

Основная функция добавления списка свойств рисунка приведена в листинге 13.18. Именно эту функцию мы и можем применять в AutoCAD 2002 и для создания, и для изменения словаря.

Листинг 13.18. Функция `ru-dwgprops-dic-make-or-add`

```
(defun ru-dwgprops-dic-make-or-add (props_list)
;|-------------------------------------------------------------------------
Пример:
(ru-dwgprops-dic-make-or-add (list '(300 . "M1:=-200") '(301 . "км")
         '(302 . "Правый угол")))
-------------------------------------------------------------------------|;
  (setq props_list
         (ru-list-union-two-assoc
           '((0 . "XRECORD")
             (100 . "AcDbXrecord")
             (1 . "DWGPROPS COOKIE")
           )
           (ru-list-union-two-assoc
             (dictsearch (namedobjdict) "DWGPROPS")
              (ru-dwgprops-test-list props_list)
          );_ end of ru-list-union-two-assoc
        );_ end of ru-list-union-two-assoc
 );_ end of setq
  (dictremove (namedobjdict) "DWGPROPS")
  (dictadd (namedobjdict) "DWGPROPS" (entmakex props_list))
  (dictsearch (namedobjdict) "DWGPROPS")
);_ end of defun
```

Для формирования корректного списка свойств применена функция `ru-dwgprops-test-list` (листинг 13.19). Корректность списка должна заключаться в том, что:

- все наши данные передаются в строковом виде;
- не должны передаваться ненужные свойства (ненужные удаляются функцией, приведенной в листинге 13.20);
- свойства с кодами 300—309 должны включать символ-разделитель =, причем только один раз;
- правая часть (после разделителя =) свойства с кодом 300 (масштаб) должна получаться преобразованием из положительного целого числа;
- правая часть (после разделителя =) свойства с кодом 301 (единица рисунка) должна передаваться только в виде мм или м.

Листинг 13.19. Функция `ru-dwgprops-test-list`

```
(defun ru-dwgprops-test-list (props_list)
  (mapcar
    (function(lambda (x) (cons (car x) (ru-dwgprops-check-property
                                    (car x) (cdr x)))))
    (ru-dwgprops-clear-nonstd props_list)
 );_ end of mapcar
);_ end of defun
```

Листинг 13.20. Функция `ru-dwgprops-clear-nonstd`

```
(defun ru-dwgprops-clear-nonstd (props_list)
;|-------------------------------------------------------------------
 Очистка списка от свойств с не нашими кодами
-------------------------------------------------------------------|;
  (vl-remove-if-not
    (function (lambda (x)
                (member (car x) (mapcar 'car
                                  (ru-dwgprops-names-by-codes-map)))
            );_ end of lambda
   );_ end of function
    props_list
 );_ end of vl-remove-if-not
);_ end of defun
```

Листинг 13.21. Функция `ru-dwgprops-check-property`

```
(defun ru-dwgprops-check-property (code value / err_string)
;|-------------------------------------------------------------------
Возвращает корректную строку
Примеры:
(ru-dwgprops-check-property 2 11111) вернет "11111"
(ru-dwgprops-check-property 304 "Свойство=значение") вернет
"Свойство 5=значение"
(ru-dwgprops-check-property 300 "Мас 1:=23.5") вернет "        М 1:=23"
```

```
(ru-dwgprops-check-property 300 "23.5") вернет "        М 1:=23"
(ru-dwgprops-check-property 301 "км") вернет "    Ед. рис. =мм"
(ru-conv-x-to-sring NIL) вернет "?"
А вот nil не передавать строкой, он в nil и превратится
-------------------------------------------------------------------------|;
  (setq err_string "")
  (if value  (if (ru-is-string value)
      (if (and (>= code 300) (< code 303))
         (setq value (ru-string-right-part value "="))
     );_ end of if
      (setq value (ru-conv-x-to-sring value))
   );_ end of if
    (setq value "?")
 );_ end of if
;; Проверка масштаба
  (cond
    ((= code 300)
     (if (not (> (atoi value) 0))
         (setq err_string "??"
          value "100")
       (setq value (itoa (ru-match-floor (atoi value))))
    );_ end of if
   )
    ((= code 301)
     (if (not (member value (list "м" "мм")))
         (setq err_string "??"
          value "мм")
    );_ end of if
   )
 );_ end of cond
 ;; присоединение =
  (cond
    ((and (>= code 300) (< code 303))
;;Принудительная замена имени свойств для трех кодов
     (strcat err_string (ru-dwgprops-rus-name-by-code code) "=" value)
   )
    (t value)
 );_ end of cond
);_ end of defun
```

Работа со свойствами через ActiveX в AutoCAD 2002

Получить список свойств можно и объектными методами. Очень важно, что таким образом в AutoCAD 2002 мы можем прочитать свойства не только активного документа, но и закрытого файла (листинг 13.22).

Листинг 13.22. Функция `ru-dwgprops-obj-get-dwgprops`

```
(defun ru-dwgprops-obj-get-dwgprops (document / dict_props_obj
safe_arr_types safe_arr_data)
```

```
  (ru-error-catch
    (function (lambda ()
       (setq dict_props_obj
              (vlax-invoke-method
                (vlax-get-property Document 'Dictionaries)
                'Item
                "DWGPROPS"
              );_ end of vlax-invoke-method
      );_ end of setq
    );_ end of lambda
   )
    nil
 );_ end of ru-error-catch
  (if dict_props_obj
    (ru-error-catch
      (function (lambda ()
         (vlax-invoke-method dict_props_obj 'GetXRecordData
           'safe_arr_types 'safe_arr_data
        );_ end of vlax-invoke-method
         (mapcar
           'cons
           (vlax-safearray->list safe_arr_types)
           (mapcar 'vlax-variant-value
             (vlax-safearray->list safe_arr_Data))
        );_ end of mapcar
      );_ end of lambda
     )
      nil
   );_ end of ru-error-catch
 );_ end of if
);_ end of defun
```

Сложность такого подхода вызвана необходимостью преобразования безопасных массивов в списки и потребностью установки ловушек для ошибок, которые могут возникнуть в любой момент. Для "поимки" ошибок разработана специальная оболочка для ловушек ошибок (листинг 13.23). Функции `ru-error-catch` передаются (с апострофом) защищаемое выражение и выражение, выполняющееся в случае ошибки (или `nil`, если ошибку просто надо предотвратить, но ничего не нужно делать).

Листинг 13.23. Функция `ru-error-catch`

```
(defun ru-error-catch (protected_expression on_error_expression /
                                                  catch_error_result)
 (setq catch_error_result (vl-catch-all-apply protected_expression))
 (if (and (vl-catch-all-error-p catch_error_result) on_error_expression)
  (apply on_error_expression
     (list (vl-catch-all-error-message catch_error_result)))
    catch_error_result
 )
)
```

Получение свойств постороннего файла

Для практического использования свойств постороннего файла разработаем дополнительную функцию (листинг 13.24), которой передается имя внешнего файла. Объект документа внешнего файла открывается с помощью COM-объекта `ObjectDBX.AxDbDocument`[1]. В функции `ru-error-catch` использован и второй аргумент — безымянная функция, выводящая сообщение об ошибке, но не прерывающая основную функцию.

Листинг 13.24. Функция `ru-dbx-props-find-all-in-file`

```
(defun ru-dbx-props-find-all-in-file (file_name / result dbx_doc props
                                                  codes)
  (ru-error-catch
    (function (lambda ()
       (setq dbx_doc
              (vla-getinterfaceobject
                (vlax-get-acad-object)
                "ObjectDBX.AxDbDocument"
              );_ end of vla-GetInterfaceObject
     );_ end of setq
      (vla-open dbx_doc file_name)
      (setq codes (mapcar 'cdr (ru-dwgprops-codes-map))
            props (ru-dwgprops-obj-get-dwgprops dbx_doc)
            names (mapcar 'car (ru-dwgprops-names-map))
     );_ end of setq
      (vlax-release-object dbx_doc)
      (mapcar 'cons
              names
              (mapcar
                (function (lambda (x)
                   (car (ru-list-massoc x props))
                );_ end of lambda
                )
                codes
             );_ end of mapcar
     );_ end of mapcar
    );_ end of lambda
   )
    (function (lambda (err_msg)
       (ru-msg-print (strcat "\nНевозможно прочитать свойства файла "
                        file_name "\n") err_msg 1)
       nil
    );_ end of lambda
   )
 );_ end of ru-error-catch
);_ end of defun
```

[1] В AutoCAD 2004 должно передаваться имя объекта `ObjectDBX.AxDbDocument.16`.

В поисках решения по работе со свойствами файлов формата AutoCAD 2004 мы нашли в Интернете[1] очень удобную библиотеку DwgPropsX.dll от Byron Blattel (CADwerx — Applications for AutoCAD). Эта библиотека, обнаруженная благодаря поиску в дискуссионных группах по продуктам фирмы Autodesk, позиционируется как "redistributal ActiveX control from Autodesk for reading drawing summary properties outside of AutoCAD"[2]. Привлекает то, что библиотека реализована в виде ActiveX (что позволяет использовать ее в любых программах), и то, что она позволяет прочитать свойства любого файла, включая файлы версии AutoCAD 2004.

Замечание

Сама фирма Autodesk для работы со свойствами предлагает использовать архив dwgpro15.zip, включающий файлы pscan.cpp, DwgProps.h и DwgProps.obj. В первом файле имеется пример программы, извлекающей свойства DWG, а остальные два предназначены для использования при разработке программ. К сожалению, такие OBJ-файлы могут применяться только при использовании компиляторов Microsoft, только для файлов версии R15, напрямую прикомпоновать DwgProps.obj к Delphi-программе нельзя, и нам пришлось писать специальную DLL на VC++ для использования функций из этой библиотеки *(см. главу 17)*. К сожалению, на момент написания этих строк фирма не позаботилась об удобном средстве для работы с файлами AutoCAD 2004 и нам приходится заниматься самодеятельностью или поиском решений третьих фирм.

Использовать библиотеку DwgPropsX.dll очень просто. Библиотеку, разумеется, нужно зарегистрировать в реестре: regsvr32 DwgPropsX.dll, после чего методы этой библиотеки доступны любым программам, использующим COM-технологии *(см. главу 15)*.

После регистрации библиотеки можно добраться до ее интерфейсов. Мы обычно это делаем в Borland Delphi путем импортирования библиотеки типов, а когда запускать Delphi лень, используем утилиту TypeExport[3] от Binh Ly, позволяющую экспортировать информацию о зарегистрированных библиотеках типов в xxx_TLB.pas, с которым потом можно удобно разобраться. Экспортировав информацию в файл DWGPROPSXLib_TLB.pas, мы получаем требуемые сведения (листинг 13.25). То, что нас интересует, выделено полужирным шрифтом в фрагментах листинга.

Листинг 13.25. Фрагменты файла DWGPROPSXLib_TLB.pas

```
const
  LIBID_DWGPROPSXLib: TGUID = '{0EE0A6F0-0859-420C-A259-94E4EF741BE5}';
  IID_IProperties: TGUID = '{5D74E9D6-6C1C-48D9-A197-05069B5CE812}';
...
  IPropertiesDisp = dispinterface
    ['{5D74E9D6-6C1C-48D9-A197-05069B5CE812}']
    procedure Load (const bstrFileName: WideString); dispid 1;
    procedure LetOff; dispid 2;
    property Title: WideString readonly dispid 3;
```

[1] **www.cadwerx.net**, файл **www.cadwerx.net/public/downloads/DwgPropsX.zip**.

[2] Приводим без перевода, так как нас смущают выражения "redistributal" и "from Autodesk". На сайте самой фирмы Autodesk нам эту библиотеку обнаружить не удалось, хотя там ей самое место.

[3] **www.techvanguards.com**.

```
    property Subject: WideString readonly dispid 4;
    property Author: WideString readonly dispid 5;
    property Comments: WideString readonly dispid 6;
    property Keywords: WideString readonly dispid 7;
    property LastSavedBy: WideString readonly dispid 8;
    property RevisionNumber: WideString readonly dispid 9;
    property HyperlinkBase: WideString readonly dispid 10;
    property Custom [index: Integer]: WideString readonly dispid 11;
    property EditingTime: TDateTime readonly dispid 12;
    property Created: TDateTime readonly dispid 13;
    property LastUpdated: TDateTime readonly dispid 14;
  end;
```

После анализа[1] этого файла мы знаем, что нам нужно использовать объект DWGPROPSX.Properties, у которого имеются все требуемые нам свойства. Для совместимости с функциями, ориентированными на работу со словарем, список свойств удобнее возвращать в виде ассоциированного списка (листинг 13.26).

Листинг 13.26. Функция `ru-dwgprops-obj-get-from-file`

```
(defun ru-dwgprops-obj-get-from-file (file_name / i result srv)
 (if (setq srv (vlax-get-or-create-object "DWGPROPSX.Properties"))
  (progn
   (vlax-invoke-method srv 'load file_name)
    (setq result (cons (cons 2 (vlax-get-property srv 'Title)) result)
        result (cons (cons 3 (vlax-get-property srv 'Subject)) result)
        result (cons (cons 4 (vlax-get-property srv 'Author)) result)
        result (cons (cons 6 (vlax-get-property srv 'Comments)) result)
        result (cons (cons 7 (vlax-get-property srv 'Keywords)) result)
            i 0)
    (repeat 10
     (setq result
      (cons
        (cons (+ 300 i) (vlax-get-property srv 'Custom i)) result)
              i (1+ i));_ end of setq
   );_ end of repeat
   );_ end of progn
    (alert "Не могу запустить DWGPROPSXLib.Properties")
 );_ end of if
  (reverse result)
);_ end of defun
```

Испытаем эту функцию на файлах различных версий. Прочитаем файл версии 2002:

```
(ru-dwgprops-obj-get-from-file "C:\\.ru\\cad\\samples\\dwg\\Сантехника
\\demo-Водомер.dwg")
```

[1] Разбираться с такими файлами читатели смогут после того, как в главе 15 мы разработаем несколько собственных ActiveX DLL.

вернет

```
((2 . "Водомерный узел № 2") (3 . " КЗКТ. Корпус 15. 3 очередь")
  (4 . "Гипроавтоагрегат. ОТЭ. Исп Зуев С.А. ГИП Корляков Н.Л.")
  (6 .
    "Дополнительный водомерный узел №2 корпуса 15 КЗКТ. Вариант с обводной линией."
 ) (7 . "КЗКТ водоснабжение корпус 15") (300 . "        М 1:=2")
  (301 . "    Ед. рис. =мм") (302 . "Тчк. вст=") (303 . "Программист=")
  (304 . "Админ=Да") (305 . "Помощник=Нет") (306 . "Чайник=Нет")
  (307 . "=") (308 . "=") (309 . "=")
)
```

Вызов

```
(ru-dwgprops-obj-get-from-file "C:\\.ru\\cad\\samples\\dwg\\2004\\Test2004.dwg")
```

вернет

```
((2 . "Проба 2004") (3 . "Проверка свойств") (4 . "Автор")
  (6 . "комментарии к рисунку") (7 . "ключевые слова")
 (300 . "Масштаб=500") (301 . "Ед. рис=м")
 (302 . "Точка вст.=Начало координат") (303 . "=") (304 . "=")
  (305 . "=") (306 . "=") (307 . "=") (308 . "=") (309 . "=")
)
```

Эта функция кажется нам более удобной, чем вызов `ObjectDbx`, и мы в дальнейшем будем применять именно функцию `ru-dwgprops-obj-get-from-file`.

Работа с *SummaryInfo* в AutoCAD 2004[1]

Напишем функцию для извлечения свойств рисунка с использованием объекта `SummaryInfo` (листинг 13.27). Функцию будем сразу писать в оптимизированном виде, с использованием ловушки ошибок, с возвратом результата в виде ассоциированного списка. Внутри мы воспользуемся функцией `vla-get-title` и подобными ей (таких функций, предназначенных для более удобного извлечения свойств, в Visual LISP много).

Листинг 13.27. Функция `ru-dwgprops-get-by-summary-info`

```
(defun ru-dwgprops-get-by-summary-info (/ summary_info result i)
(ru-error-catch
    (function (lambda ()
       (setq summary_info
       (vlax-get-property
        ru-obj-get-active-document) 'SummaryInfo
      );_ end of vlax-get-property
      result
      (cons (cons 7 (vla-get-keywords summary_info))
       (cons (cons 6 (vla-get-comments summary_info))
```

[1] Распространяется и на AutoCAD 2005.

```
        (cons (cons 4 (vla-get-author summary_info))
         (cons (cons 3 (vla-get-subject summary_info))
          (cons (cons 2 (vla-get-title summary_info)) result)
         );_ end of cons
        );_ end of cons
       );_ end of cons
      );_ end of cons
      i 0
  );_ end of setq
   (repeat (vlax-invoke-method summary_info 'NumCustomInfo)
    (vla-getcustombyindex summary_info i 'CustomName 'CustomValue)
    (setq result
          (cons (cons (+ 300 i)
       (strcat CustomName "=" CustomValue)) result)
      i   (1+ i)
   );_ end of setq
  );_ end of repeat
   (reverse result)
 );_ end of lambda
 )
 (function (lambda (err_msg)
   (ru-msg-print
    (strcat "\nRU-DWGPROPS-GET-BY-SUMMARY-INFO:"
         "Ошибка чтения свойств\n") err_msg 1
  );_ end of ru-msg-print
   nil
 );_ end of lambda
 )
);_ end of ru-error-catch
);_ end of defun
```

Изменение свойств будем выполнять с помощью функции `ru-dwgprops-set-summary-info` (листинг 13.28).

Листинг 13.28. Функция `ru-dwgprops-set-summary-info`

```
(defun ru-dwgprops-set-summary-info (props_list / summary_info result i)
  (ru-error-catch
    (function (lambda ()
       (setq props_list (ru-dwgprops-test-list props_list)
             summary_info (vlax-get-property (ru-obj-get-active-document)
                          'SummaryInfo);_ end of vlax-get-property
      );_ end of setq
       (vla-put-title summary_info (cdr (assoc 2 props_list)))
       (vla-put-subject summary_info (cdr (assoc 3 props_list)))
       (vla-put-author summary_info (cdr (assoc 4 props_list)))
       (vla-put-comments summary_info (cdr (assoc 6 props_list)))
       (vla-put-keywords summary_info (cdr (assoc 7 props_list)))
       (setq i 0)
```

```
      (repeat 10
        (if (setq custom (cdr (assoc (+ 300 i) props_list)))
          (if (>= i (vlax-invoke-method summary_info 'NumCustomInfo))
            (vla-addcustominfo summary_info
              (ru-string-left-part custom "=")
              (ru-string-right-part custom "=")
            );_ end of vla-addcustominfo
            (vla-setcustombyindex summary_info i
              (ru-string-left-part custom "=")
              (ru-string-right-part custom "=")
            );_ end of vla-setcustombyindex
          );_ end of if
        );_ end of if
        (setq i (1+ i))
      );_ end of repeat
      (ru-dwgprops-get-by-summary-info)
    );_ end of lambda
   )
    (function (lambda (err_msg)
      (ru-msg-print
        (strcat "\nRU-DWGPROPS-SET-SUMMARY-INFO:"
                "Ошибка установки свойств\n") err_msg  1
      );_ end of ru-msg-print
      nil
    );_ end of lambda
   )
 );_ end of ru-error-catch
)
```

Кто виноват?

Многократное тестирование этих функций показывает, что свойства объекта `SummaryInfo` программным путем изменяются и читаются верно. Произведенные изменения правильно отображаются в диалоговом окне команды DWGPROPS (СВОЙСТВАРИС), но изменения, сделанные вручную в этом окне, не отражаются в свойствах `SummaryInfo`.

Следовательно, виновником наших бед является команда DWGPROPS (СВОЙСТВАРИС), вернее, программисты, не предусмотревшие правильную обработку результатов ее работы.

Что делать?

Попробуем переопределить команду DWGPROPS (СВОЙСТВАРИС). В *главе 15* мы разработаем диалоговое окно, позволяющее отображать и редактировать двойной список `Переменная=Значение`. Мы легко можем заменить им стандартное диалоговое окно, причем возможно отображать множество различных свойств, не только стандартных. Если переопределение команды будет произведено сразу после открытия файла, то пользователь будет видеть только наше окно, а мы уж сумеем правильно сохранить изменения свойств. Как будет выглядеть наше диалоговое окно, показано на рис. 13.1.

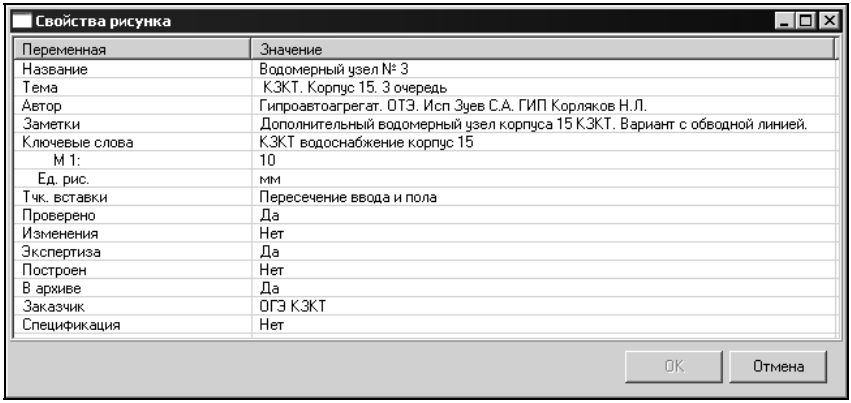


Рис. 13.1. Диалоговое окно-заместитель редактора свойств рисунка

В результате редактирования в таком диалоговом окне возвращается список из двух ассоциированных списков. Первый подсписок — пара "переменная-значение" выбранной строки, а второй ассоциированный список — пары названий переменных и измененных значений (если не запрещен режим редактирования). Первый подсписок нас не интересует, а второй подсписок мы должны разобрать, причем с учетом, что при щелчке по заголовкам колонок изменяется порядок сортировки отображаемого списка. Из-за этого мы не можем просто использовать порядковые номера в исходном списке, а должны произвести поиск пар "код-значение".

Правильность любых значений переменных мы можем проконтролировать при закрытии диалогового окна.

Прежде чем мы примем окончательное решение, разработаем универсальные функции для работы со свойствами в любой версии системы AutoCAD.

Читать список свойств мы будем с помощью функции `ru-dwgprops-get` (листинг 13.29).

Листинг 13.29. Функция `ru-dwgprops-get`

```
(defun ru-dwgprops-get ()
  (if (> (ru-acad-ver) 15)
    (ru-dwgprops-test-list (ru-dwgprops-get-by-summary-info))
    (ru-dwgprops-test-list (dictsearch (namedobjdict) "DWGPROPS"))
 );_ end of if
)
```

Записывать список свойств мы будем с помощью функции `ru-dwgprops-set` (листинг 13.30).

Листинг 13.30. Функция `ru-dwgprops-set`

```
(defun ru-dwgprops-set (dwg_props)
  (if (> (ru-acad-ver) 15)
    (ru-dwgprops-set-summary-info dwg_props)
```

```
    (ru-dwgprops-dic-make-or-add dwg_props)
 );_ end of if
)
```

Редактировать список свойств активного рисунка мы будем с помощью функции `ru-dwgprops-edit` (листинг 13.31).

Листинг 13.31. Функция `ru-dwgprops-edit`

```
(defun ru-dwgprops-edit (/dwg_props list_codes_names list_names
 list_values new_list result code name value x)
  (setq dwg_props (ru-dwgprops-get)
        list_codes_names
          (mapcar (function (lambda (x) (cons (car x) (if (< (car x) 300)
          (ru-dwgprops-rus-name-by-code (car x))
          (ru-string-left-part (cdr x) "="))))))
          dwg_props);_ end of mapcar
        list_names
          (mapcar 'cdr list_codes_names)
        list_codes_names (ru-list-reverse-assoc list_codes_names)
        list_values
          (mapcar (function (lambda (x) (if (< (car x) 300) (cdr x)
                    (ru-string-right-part (cdr x) "="))))
                              dwg_props);_ end of mapcar
 );_ end of setq
  (if (setq new_list (ru-dlg-double-list "Свойства рисунка"
                       "Переменная" "Значение" list_names list_values
                       nil
                     );_ end of ru-dlg-double-list
    );_ end of setq
    (progn
      (foreach x (cadr new_list)
        (setq name (car x)
              code (cdr (assoc name list_codes_names))
        );_ end of setq
        (if (< code 300)
          (setq value (cdr x))
          (setq value (strcat name "=" (cdr x)))
        );_ end of if
        (setq
          result (cons (cons code value)
                       result
                 );_ end of cons
        );_ end of setq
      );_ end of foreach
    );_ end of progn
 );_ end of if
  (reverse result)
)
```

В результате работы этой функцией мы имеем список свойств в стандартном формате. Но нам еще нужно протестировать результаты ручного редактирования и только при правильных данных записать свойства в рисунок. Для этого предназначена функция `ru-dwgprops-edit-and-set` (листинг 13.32).

Листинг 13.32. Функция `ru-dwgprops-edit-and-set`

```
(defun ru-dwgprops-edit-and-set (/ dwg_props test)
  (while (and (not test) (setq dwg_props (ru-dwgprops-edit)))
    (if dwg_props
      (if (setq test (ru-dwgprops-test-summary dwg_props))
        (ru-dwgprops-set dwg_props)
     );_ end of if
   );_ end of if
 );_ end of while
);_ end of defun
```

Листинг 13.33. Функция `ru-dwgprops-test-summary`

```
(defun ru-dwgprops-test-summary (dwg_props / new_std_dwg_props error_msg
                                             scale unit_name)
  (setq error_msg ""
        new_std_dwg_props
         (list (atoi (ru-string-right-part
                (ru-ent-dxf-code-data 300 dwg_props) "="))
               (ru-string-right-part (ru-ent-dxf-code-data 301 dwg_props)
                "="));_ end of list
        scale     (nth 0 new_std_dwg_props)
        unit_name (nth 1 new_std_dwg_props)
 );_ end of setq
  (if (not (> scale 0))
    (setq error_msg (strcat error_msg "Введен неверный масштаб чертежа!"
         "\nДопустимы только ПОЛОЖИТЕЛЬНЫЕ ЧИСЛА\n"));_ setq
 );_ if
  (if (not (ru-is-int scale))
    (setq error_msg (strcat error_msg "Введен неверный масштаб чертежа"
         "\nДопустимы только ЦЕЛЫЕ числа\n"));_ setq
 );_ if
  (if (not (member unit_name (list "м" "мм")))
  (setq error_msg (strcat error_msg "Введены неверные единицы рисунка: '"
         unit_name "'\nДопустимые: 'м' или 'мм'\n")));_ if
  (if (/= error_msg "")
    (progn
      (ru-msg-alert (strcat error_msg "\nПовторите ввод!"))
      nil
   );_ progn
    t
 );_ if
);_ end of defun
```

Теперь мы можем написать заменитель команды DWGPROPS (СВОЙСТВАРИС) (листинг 13.34).

Листинг 13.34. Функция C:DWGPROPS

```
(defun C:DWGPROPS ()
(ru-dwgprops-edit-and-set)
(princ)
)[1]
```

Из четырех рассматривавшихся вариантов мы примем последний — переопределение стандартной команды. Переопределение команды лучше всего делать при инициализации рисунка, а определение функции C:DWGPROPS удобно поместить в один файл с функцией инициализации.

Использование свойств рисунка при инициализации системы

Функция инициализации системы, запускаемая при открытии каждого рисунка, приведена в листинге 13.35. В ней имеются несколько вызовов пока неизвестных функций, выделенных полужирным шрифтом, а саму функцию можно рассматривать как план первоочередных работ. Малозначительные участки кода мы опускаем для лучшего понимания общей структуры работы.

Листинг 13.35. Функция ru-init-start-rucad

```
(defun C:DWGPROPS ()
(ru-dwgprops-edit-and-set)
(princ)
)
(defun ru-init-start-rucad (/ dwg_ini dwg_name dwg_props_from_ini
                                                lst_std_dwg_props)
;|-------------------------------------------------------------------------
Инициализация системы
-------------------------------------------------------------------------|;
 (setvar "CMDECHO" 0)
     (setq dwg_name (ru-file-full-dwgname)
      dwg_ini (ru-file-set-ext dwg_name ".ini")
    )
;|-------------------------------------------------------------------------
Отмена стандартной команды DWGPROPS
-------------------------------------------------------------------------|;
 (command "_.UNDEFINE" "DWGPROPS")
```

[1] Для русской версии системы AutoCAD можно добавить строку:
(defun C:СВОЙСТВАРИС () (C:DWGPROPS)).

```
;|-------------------------------------------------------------------------
Надо знать, новый ли рисунок и установлены ли у него свойства
-------------------------------------------------------------------------|;
  (if (not (ru-dwgprops-test (ru-dwgprops-get)))
    (progn
;|-------------------------------------------------------------------------
Надо знать, новый ли рисунок и установлены ли у него свойства
При этом:
а) Возможен вариант, что файл еще не открывался, но у него есть dwg_ini,
в котором заданы параметры (пример - файл типового проекта)
б) может быть словарь, но в нем не установлены масштаб и единицы
в) может быть вновь созданный файл
Свойства по умолчанию для нового рисунка мы можем прочитать из файла шаблонов
свойств DwgPropsIni.templ
-------------------------------------------------------------------------|;
     (if (setq dwg_props_from_ini
                (ru-dwgprops-read-from-ini dwg_name
                  (ru-file-template "DwgPropsIni.templ")
                );_ end of ru-dwgprops-read-from-ini
       );_ end of setq
        (ru-dwgprops-set-by-ini dwg_props_from_ini)
    );_ end of if
;|-------------------------------------------------------------------------
Теперь мы имеем установленную заготовку свойств по умолчанию. На следующем шаге
пытаемся прочитать свойства из INI-файла рисунка
-------------------------------------------------------------------------|;
     (if (/= (ru-ini-read dwg_ini "AcadDWGCustom1" "Value" "X") "X")
       (if (setq dwg_props_from_ini
                 (ru-dwgprops-read-from-ini dwg_name dwg_ini))
         (progn
           (ru-dwgprops-set-by-ini dwg_props_from_ini)
           (ru-dwgprops-erase-section dwg_ini)
         );_ end of progn
       );_ end of if
    );_ end of if
;|-------------------------------------------------------------------------
Так как функция ru-init-start-rucad запускается при открытии каждого рисунка, и при
прочтении свойств из файла dwg_ini в нем полностью удаляются секции с описанием
свойств (в дальнейшем свойства сохраняются в самом рисунке), то файл dwg_ini можно
рассматривать как временное хранилище обновленных свойств.
После установки свойств выводим стандартный диалог свойств - на самом деле будет
выведен наш, нестандартный диалог, хотя вызывается имя команды с точкой в префиксе.
-------------------------------------------------------------------------|;
      (C:DWGPROPS)
   );_ end of progn
 );_ end of if
;|-------------------------------------------------------------------------
 Перечитываем свойства, возможно, измененные в диалоге
-------------------------------------------------------------------------|;
  (setq lst_std_dwg_props (ru-dwgprops-get-std))
```

```
;|-------------------------------------------------------------------------
 Настраиваем рисунок на масштаб и единицы
-------------------------------------------------------------------------|;
(ru-init-setup-dwg-by-scale (nth 0 lst_std_dwg_props)
                            (nth 1 lst_std_dwg_props))
 (ru-menu-ru-load)
 (princ)
);_ end of defun
```

Пояснение

В этой функции мы пытаемся прочитать свойства из INI-файла рисунка. Зачем это, если мы храним их внутри файла? Дело в том, что у нас появятся программы, записывающие типовые проектные решения в файлы. Свойства таких файлов мы задаем в момент записи, но записать их в закрытый файл мы не можем. Однако мы можем записать их в INI-файл рисунка и прочитать свойства из этого файла. Так как при загрузке типового проектного решения будет выполняться эта же функция инициализации, чтение INI-файла предусмотрено здесь. После прочтения переменных все секции свойств уничтожаются. Подробности см. в *главе 33*.

Визуально работа системы при создании нового рисунка (вернее, не нового, а не побывавшего еще "в лапах" системы ruCAD) начинается с вывода диалогового окна редактирования свойств. После закрытия этого окна (а оно может вызваться пользователем не только при старте) необходимо отследить изменения свойств. Для этого мы должны разработать командный реактор, контролирующий выполнение команды DWGPROPS (СВОЙСТВАРИС).

Командный реактор DWGPROPS

Реактор команд типа `:VLR-command-reactor` создается с помощью функции

```
(vlr-command-reactor <данные> <реакции>)
```

Такой реактор должен реагировать на события:

- `:vlr-commandWillStart` — вызов команды AutoCAD;
- `:vlr-commandEnded` — завершение команды;
- `:vlr-commandCancelled` — прерывание команды.

В нашем случае при вызове команды должны запоминаться значения базовых свойств, при завершении команды (кнопка **OK** диалогового окна) должны проверяться введенные значения и, возможно, требоваться исправления, при прерывании команды можно ничего не делать, т. к. прежние свойства не будут изменены самой системой AutoCAD.

Реакторы должны быть загружены в составе библиотеки функций, но не должны вызываться напрямую. Нам необходимо проверять наличие такого реактора, иначе при многократной загрузке библиотеки будет происходить многократная обработка одного события.

Итак, пишем функции-реакторы. Реакторы мы соберем в файл ru-vlr.lsp. Этот файл скомпилируется в общую библиотеку, но в нем, в отличие от других файлов, будет "внефункциональный" участок кода, выполняющийся в момент загрузки (листинг 13.36).

Листинг 13.36. Создание реактора

```
;|-------------------------------------------------------------------------
Внефункциональный участок файла ru-vlr.lsp
Для предотвращения повторного создания реактора при перезагрузке этого файла просто
удаляем реакторы команд. Наш реактор команд сохраняется в глобальной переменной
*ru_command_reactor*
-------------------------------------------------------------------------|;
 (if *ru_command_reactor*
  (progn
      (setq *ru_command_reactor* nil)
      (vlr-remove-all :vlr-command-reactor)
 );_ end of progn
);_ end of if
;;; Если нашего реактора еще нет, создаем его
(if (not *ru_command_reactor*)
    (setq *ru_command_reactor*
     (vlr-command-reactor
          nil;; Никакие данные с реактором не ассоциируются
           '(;; Создаем список событий, используемых реактором
             ;; Событие вызова команды
              (:vlr-commandWillStart . ru-vlr-start-command)
             ;; Событие завершения команды
               (:vlr-commandEnded . ru-vlr-end-command)
           )
     )
   );_ end of setq
);_ end of if
```

Итак, мы создали реактор команд, который будет реагировать на вызов и завершение любой команды AutoCAD. При вызове команды будет вызываться функция `ru-vlr-start-command` (листинг 13.37), а при завершении — `ru-vlr-end-command` (листинг 13.38).

Листинг 13.37. Функция `ru-vlr-start-command`

```
(defun ru-vlr-start-command (reactor command_info / command_name)
;|-------------------------------------------------------------------------
Реактор вызова команды
AutoCAD сам передает функции два параметра:
reactor — объект действующего реактора, например #<VLR-Command-Reactor>
command_info — список вызванных команд, в нем будет одна команда, например
("DWGPROPS")
-------------------------------------------------------------------------|;
;;; Извлекаем имя активной команды
(setq command_name (nth 0 command_info))
;;; В зависимости от имени команды выполняем какие-то действия.
;;; Проверку делаем через cond (хотя пока всего одно условие), чтобы
;;; в дальнейшем можно было добавлять другие команды
  (cond
    ((= command_name "DWGPROPS")
```

```
;;; Запоминаем в глобальной переменной список базовых свойств
;;; Он будет иметь вид наподобие(100 "мм")
      (setq *ru_std_dwg_props* (ru-dwgprops-get-std))
   )
 );_ end of cond
(princ)
);_ end of defun
```

Листинг 13.38. Функция ru-vlr-end-command

```
(defun ru-vlr-end-command (reactor command_info / command_name
                  new_std_dwg_props error_msg scale unit_name)
;|-------------------------------------------------------------------------
Реактор завершения команды
Аргументы этой функции такие же, как и у ru-vlr-start-command
-------------------------------------------------------------------------|;
(setq command_name (nth 0 command_info)
      error_msg "")
(cond
    ((= command_name "DWGPROPS")
;|-------------------------------------------------------------------------
Надо проверить правильность заполнения свойств и, если свойства изменены, сохранить
в словаре и переинициализировать рисунок
-------------------------------------------------------------------------;
     (ru-dwgprops-test-input)
   )
 ) (princ)
);_ end of defun
```

Теперь осталось написать еще несколько функций для работы со свойствами рисунка (листинги 13.39 и 13.40).

Листинг 13.39. Функция ru-dwgprops-set-std

```
(defun ru-dwgprops-set-std (scale unit_name)
;|-------------------------------------------------------------------------
Запись в словарь масштаба и единиц рисунка
Пример:
Command: (ru-dwgprops-set-std 25 "мм")
-------------------------------------------------------------------------|;
  (setq *ru_dwg_scale*     scale
        *ru_dwg_units*     unit_name
        *ru_std_dwg_props* (list scale unit_name)
 );_ end of setq
  (ru-dwgprops-set
    (ru-list-union-two-assoc (ru-dwgprops-get)
      (ru-dwgprops-test-list
        (list (cons 300 (itoa (ru-match-floor scale)))
              (cons 301 unit_name)
       );_ end of list
     );_ end of ru-dwgprops-test-list
```

```
   );_ end of ru-list-union-two-assoc
 );_ end of ru-dwgprops-set
)
```

Листинг 13.40. Функция ru-dwgprops-get-std

```
(defun ru-dwgprops-get-std (/ props_list)
 (if (ru-dwgprops-test(setq dwg_props (ru-dwgprops-get)))
    (list (atoi (ru-string-right-part
          (ru-ent-dxf-code-data 300 dwg_props) "="))
          (ru-string-right-part (ru-ent-dxf-code-data 301 dwg_props) "=")
   );_ end of list
   (list 100 "мм")
 )
)
```

Вопрос на засыпку

А нужен ли теперь командный реактор? Мы переопределили стандартную *команду* DWGPROPS, заменив собственной *функцией* `C:DWGPROPS`, которая только на клавиатуре набирается так же, как команда, но сделали это так хитро, что даже вызов _.DWGPROPS (с префиксами) выводит наше диалоговое окно, а не стандартное.

Фактически реактор теперь не будет работать, но он останется на случай, если хитроумный пользователь вздумает восстановить стандартную команду.

Теперь мы умеем очень много — читать и записывать любые свойства рисунка, выводимые диалоговым окном команды DWGPROPS, и контролировать ввод данных пользователем. Наши функции разработаны "на вырост", т. е. в них легко добавить дополнительные пользовательские свойства. Приведенные функции можно еще совершенствовать, например, принудительно заставлять пользователя вводить реквизиты рисунка, увязывать данные с основной надписью, дублировать их в файлах или даже помещать информацию в удаленную базу данных. Но такие приемы мы оставим для рассмотрения в другой книге.

Обеспечение пропорций

После установки масштаба и единиц мы можем поддерживать систему пропорциональности. Для реализации задуманного нам потребуется довольно много простеньких функций, обеспечивающих вычисление требуемых параметров при рисовании. Заодно напишем несколько функций преобразования (листинги 13.41—13.61), чтобы потом не возвращаться к этой теме.

Листинг 13.41. Функция ru-conv-bool-to-int

```
 (defun ru-conv-bool-to-int (val)
;|-------------------------------------------------------------------------
Преобразование логического аргумента в целое число
Если передано значение не nil, возвращает 1, иначе 0.
-------------------------------------------------------------------------|;
(if val 1 0)
);_ end of defun
```

Листинг 13.42. Функция `ru-conv-value-to-wordbool`

```
(defun ru-conv-value-to-wordbool (value / type-val)
;|-------------------------------------------------------------------------
 Преобразование значения любого типа в 1 или 0 для совместимости с COM-серверами
-------------------------------------------------------------------------|;
(cond
    ((= (setq type-val (type value)) 'INT) value)
    ((= type-val 'REAL) (if (/= value 0.0) 1 0))
    ((= type-val 'STR) (if (ru-conv-str-to-bool value) 1 0))
    ((= type-val 'variant)
       (if (= (vlax-variant-value result) :vlax-true) 1 0))
    (T (if value 1 0))
 );_ end of cond
);_ end of defun
```

Листинг 13.43. Функция `ru-conv-value-to-bool`

```
 (defun ru-conv-value-to-bool (value / type-val)
;|-------------------------------------------------------------------------
Преобразование значения любого типа в T или nil
-------------------------------------------------------------------------|;
(cond
    ((= (setq type-val (type value)) 'INT)(/= value 0))
    ((= type-val 'REAL) (/= value 0.0))
    ((= type-val 'STR) (ru-conv-str-to-bool value))
    ((= type-val 'variant)
      (= (vlax-variant-value result) :vlax-true)
    )
     (t  value)
 )
)
```

Листинг 13.44. Функция `ru-conv-bool-to-str`

```
(defun ru-conv-bool-to-str (val)
;|-------------------------------------------------------------------------
Преобразование логического значения в строку. Потребуется для сохранения в файлах
настроек
-------------------------------------------------------------------------|;
  (if val "1" "0")
);_ end of defun
```

Листинг 13.45. Функция `ru-conv-str-to-bool`

```
 (defun ru-conv-str-to-bool (s)
;|-------------------------------------------------------------------------
Преобразование строки в логическое значение
Используем несколько вариантов строковой "истины"
-------------------------------------------------------------------------|;
(not (null
 (member (strcase s) '("1" "Y" "YES" "ДА" "Д" "T" "TRUE" ".T."))))
);_ end of defun
```

Листинг 13.46. Функция ru-conv-deg-to-rad

```
 (defun ru-conv-deg-to-rad (deg_ang)
;|-------------------------------------------------------------------------
Преобразование углов из градусов в радианы
-------------------------------------------------------------------------|;
  (* pi (/ deg_ang 180.0))
);_ end of defun
```

Листинг 13.47. Функция ru-conv-rad-to-deg

```
(defun ru-conv-rad-to-deg (rad_ang)
;|-------------------------------------------------------------------------
Преобразование углов из радиан в градусы
-------------------------------------------------------------------------|;
  (* 180.0 (/ rad_ang pi))
);_ end of defun
```

Листинг 13.48. Функция ru-is-int

```
(defun ru-is-int (value)
;|-------------------------------------------------------------------------
Проверка: Аргумент целое число?
-------------------------------------------------------------------------|;
  (= (type value) 'INT)
)
```

Листинг 13.49. Функция ru-is-real

```
(defun ru-is-real (value)
;|-------------------------------------------------------------------------
Проверка: Аргумент вещественное число?
-------------------------------------------------------------------------|;
 (= (type value) 'REAL)
)
```

Листинг 13.50. Функция ru-is-string

```
(defun ru-is-string (value)
;|-------------------------------------------------------------------------
Проверка: Аргумент строка?
-------------------------------------------------------------------------|;
  (= (type value) 'STR)
)
```

Листинг 13.51. Функция ru-is-paper-space

```
(defun ru-is-paper-space ()
;|-------------------------------------------------------------------------
Проверка: Текущее пространство листа?
-------------------------------------------------------------------------|;
(and (= (getvar "tilemode") 0) (= (getvar "cvport") 1))
)
```

Листинг 13.52. Функция ru-unit-name

```
(defun ru-unit-name ()
;|-----------------------------------------------------------------------
 Возвращает наименование единицы рисунка "м" или "мм"
-----------------------------------------------------------------------|;
(cond (*ru_dwg_units*) ("мм"))
;;; Вообще-то это значение при настройке сохраняется в системной
;;; переменной INSUNITS, и мы могли бы прочитать оттуда, но
;;; мы так не делаем, потому что пользователь имеет возможности
;;; изменить переменную вручную
;;; (if (= (getvar "INSUNITS") 4) "мм" "м")
);_ end of defun
```

Листинг 13.53. Функция _ru-unit-is-millimeter

```
(defun _ru-unit-is-millimeter ()
;|-----------------------------------------------------------------------
Проверка: Единицы рисунка миллиметры?
-----------------------------------------------------------------------|;
    (= (strcase (cond ((ru-unit-name))
                      (""))
               );_ end of cond
      );_ end of strcase
       "мм"
   );_ end of =
);_ end of defun
```

Листинг 13.54. Функция ru-conv-unit-to-meter

```
(defun ru-conv-unit-to-meter (value)
;|-----------------------------------------------------------------------
 Перевод числа из единиц рисунка в метры
-----------------------------------------------------------------------|;
  (if (or (ru-is-paper-space) (_ru-unit-is-millimeter))
    (* value 0.001);;В листе всегда ММ, 1мм=0.001 м
    value
 );_ end of if
)
```

Листинг 13.55. Функция ru-conv-meter-to-unit

```
(defun ru-conv-meter-to-unit (value)
;|-----------------------------------------------------------------------
Перевод числа, заданного в метрах, в единицы рисунка
-----------------------------------------------------------------------|;
(if (or (ru-is-paper-space) (_ru-unit-is-millimeter))
    (* value 1000.0)
    value
 );_ end of if
);_ end of defun
```

Листинг 13.56. Функция ru-conv-unit-to-millimeter

```
(defun ru-conv-unit-to-millimeter (value)
;|-------------------------------------------------------------------------
Перевод единиц рисунка в миллиметры
-------------------------------------------------------------------------|;
(if (or (ru-is-paper-space) (_ru-unit-is-millimeter))
      value
      (* value 1000.0)
 );_ end of if
)
```

Листинг 13.57. Функция ru-conv-millimeter-to-unit

```
 (defun ru-conv-millimeter-to-unit (value)
;|-------------------------------------------------------------------------
Перевод числа, заданного в миллиметрах, в единицы рисунка
-------------------------------------------------------------------------|;
 (if (or (ru-is-paper-space) (_ru-unit-is-millimeter))
    value
    (* value 0.001)
);_ end of if
);_ end of defun
```

Функция `ru-conv-millimeter-in-paper-to-unit` (листинг 13.58) является основной в системе поддержки пропорциональности. Именно эта функция переводит число, заданное в *миллиметрах на бумаге* (а именно так нормируются размеры масштабируемых символов), в *единицы рисунка* AutoCAD. Функция учитывает наименование единиц (*мм* или *м*), масштаб *чертежа* и текущее пространство.

Например, если потребуется написать текст высотой на бумаге 2.5 мм, то нужно передать в соответствующую функцию, рисующую текст, его высоту в виде результата функции `(ru-conv-millimeter-in-paper-to-unit 2.5)`.

Листинг 13.58. Функция ru-conv-millimeter-in-paper-to-unit

```
(defun ru-conv-millimeter-in-paper-to-unit (value)
;|-------------------------------------------------------------------------
Перевод числа, заданного в миллиметрах НА БУМАГЕ, в единицы рисунка
-------------------------------------------------------------------------|;
 (* (ru-conv-millimeter-to-unit value)
       (cond ((ru-scale-current-space))
             (1.0)
      );_ end of cond
   );_ end of *
);_ end of defun
```

Другие функции системы поддержки пропорциональности приведены в листингах 13.59—13.61.

Листинг 13.59. Функция ru-scale-model

```
(defun ru-scale-model ()
;|-----------------------------------------------------------------------------
Возвращает масштаб чертежа в пространстве модели
-----------------------------------------------------------------------------|;
(cond (*ru_dwg_scale*) (1.0))
);_ end of defun
```

Листинг 13.60. Функция ru-scale-current-space

```
(defun ru-scale-current-space ()
;|-----------------------------------------------------------------------------
Возвращает масштаб чертежа в текущем пространстве
-----------------------------------------------------------------------------|;
  (if (ru-is-paper-space) 1.0 (ru-scale-model))
)
```

Листинг 13.61. Функция ru-conv-unit-to-millimeter-in-paper

```
(defun ru-conv-unit-to-millimeter-in-paper (value)
;|-----------------------------------------------------------------------------
Преобразование числа, заданного в единицах рисунка, в миллиметры на бумаге.
Учитывается масштаб, пространство и наименование единиц
-----------------------------------------------------------------------------|;
 (/ (ru-conv-unit-to-millimeter value) (ru-scale-current-space))
)
```

Настройки системы на масштаб

Особо выделим функцию, выполняемую при создании рисунка или изменении основных свойств. В ней производятся все настройки на заданный масштаб и единицы рисунка. Именно эта функция позволяет практически обойтись без шаблонов, т. к. значения *всех системных переменных* устанавливаются в соответствии со спецификой строительного черчения и основными настройками. В листинге 13.62 мы ради экономии места не приводим установку *всех* системных переменных, покажем только важнейшие, прямо зависящие от масштаба и единиц.

Листинг 13.62. Функция ru-init-setup-dwg-by-scale

```
(defun ru-init-setup-dwg-by-scale (scale unit_name / tmp)
;|-----------------------------------------------------------------------------
 Настройка рисунка на заданный масштаб и единицы
 Аргументы:
   Scale  - масштаб
   unit_name - наименование единиц
-----------------------------------------------------------------------------|;
;;; Сохраняем параметры в словаре рисунка
(ru-dwgprops-set-std scale unit_name)
```

```
;;; т. к. масштаб требуется очень часто, сохраняем в глобальной
(setq *ru_dwg_scale* scale *ru_dwg_units* unit_name)
(if (= unit_name "мм")
  (progn
    (setvar "INSUNITS" 4) (setvar "DIMAPOST" "м")
;|
 Прячем сюда единицы, т. к. в строительных чертежах альтернативные размеры
не показываются
|;
    (setvar "DIMALTF" 0.001) (setvar "DIMADEC" 3)
;|Число десятичных знаков для основных размерных единиц.|;
    (setvar "DIMDEC" 0)
;|
 Слишком "точные" размеры для чертежей зданий могут быть вредны.
 Дадим возможность управлять точностью
|;
    (setvar "DIMRND"
      (atof (ru-ini-read-default nil "Setup" "DimRoundMM" "5.0")))
    (setq tmp (atof (ru-ini-read-default nil "Setup" "SnapMM" "50")))
;| Устанавливаем шаговую привязку, удобную для строительной части |;
   (setvar "SNAPUNIT" (list tmp tmp))
   (setvar "AXISUNIT" (list (* 10 tmp) (* 10 tmp)))
   (setvar "LUPREC" 0)
 );_ end of progn
 (progn
    (setvar "INSUNITS" 6) (setvar "DIMAPOST" "мм")
    (setvar "DIMALTF" 1000.0) (setvar "DIMADEC" 0)
    (setvar "DIMDEC" 3)
    (setvar "DIMRND"
    (atof (ru-ini-read-default nil "Setup" "DimRoundM" "0.001")))
    (setq tmp (atof (ru-ini-read-default nil "Setup" "SnapM" "1.0")))
    (setvar "SNAPUNIT" (list tmp tmp))
    (setvar "AXISUNIT" (list (* 10 tmp) (* 10 tmp)))
    (setvar "LUPREC" 2)
 );_ end of progn
);_ end of if
 (setvar "LTSCALE" (* 1 (ru-scale-model)))
 (setvar "DIMSCALE" (ru-scale-model))
 (setvar "PSVPSCALE" (ru-scale-model))
 (setvar "PDSIZE" (ru-conv-millimeter-in-paper-to-unit 1.5))
  (setq tmp (ru-ini-read (ru-file-ini "default.ini") "Setup" "TextHeight"
              "2.5"))
  (setvar "DIMTXT" (atof tmp))
  (setvar "TEXTSIZE" (ru-conv-millimeter-in-paper-to-unit (atof tmp)))
;|
Далее мы можем установить значения всех системных переменных, не зависящих напрямую
от пропорций рисунка. Приводим только несколько примеров
|;
(setvar "ANGBASE" 0.0) (setvar "ANGDIR" 0) (setvar "APBOX" 1)
(setvar "ATTDIA" 1) (setvar "ATTMODE" 1) (setvar "ATTREQ" 1)
```

```
(setvar "AUNITS" 1) (setvar "AUPREC" 4) (setvar "BLIPMODE" 0)
(setvar "CMDDIA" 1) (setvar "CMDECHO" 0) (setvar "COORDS" 2)
(setvar "DIMASSOC" 2) (setvar "DIMASZ" 0.1) (setvar "DIMAUNIT" 1)
;| Далее показываем только важнейшие переменные|;
(setvar "DIMLFAC" 1) (setvar "EXPLMODE" 0)(setvar "FILEDIA" 1)
(setvar "HIGHLIGHT" 1) (setvar "LIMCHECK" 0)(setvar "LISPINIT" 0)
(setvar "LUNITS" 2)(setvar "MEASUREMENT" 1) (setvar "MENUCTL" 1)
(setvar "MENUECHO" 0);|Вывод о сочетании MENUECHO и ^P в главе 8|;
(setvar "PLINEGEN" 1)
;|
Создаем текстовые стили
 У нас есть обязательный стиль для "лохматых" типов линий. Файл шрифта для этого
стиля мы задаем жестко
|;
 (ru-text-set-style "RU_LINE" "TXT.SHX" 0 1 0)
;|
 А вот файл шрифта для стандартного текста можно изменить в файле настроек
|;
 (ru-text-set-style "RU_CAD"
  (ru-ini-read (ru-file-ini "DEFAULT.INI")
      "Styles" "Standard.TxtStyleFile" "rucad.shx")
    0 (atof (ru-ini-read (ru-file-ini "DEFAULT.INI")
      "Styles" "Standard.TxtStyleWidth" "1.0"))
    (atof (ru-ini-read (ru-file-ini "DEFAULT.INI")
      "Styles" "Standard.TxtStyleAngle" "0.0"))
 );_ end of ru-text-set-style
  (setvar "DIMTXSTY" "RU_CAD")
  (princ)
);_ end of defun
```

В дальнейшем жизнь заставит нас отказаться от установки системных переменных в *этой* функции с помощью функции `setvar`. Нам придется разработать специальную функцию `ru-var-set-sysvars`, устанавливающую требуемые значения переменных объектными методами. Но пока мы пребываем в блаженном неведении о грозящих нам опасностях.

В функции `ru-init-setup-dwg-by-scale` встретились незнакомые пока функции `ru-ini-read`, `ru-ini-read-default` и `ru-text-set-style`. Функция чтения переменной из INI-файлов `ru-ini-read` рассматривается в нескольких вариантах в *главах 14* и *15*. Две других функции мы приведем здесь (листинги 13.63 и 13.64).

Листинг 13.63. Функция `ru-ini-read-default`

```
(defun ru-ini-read-default (ini_name section var default / result)
;|-------------------------------------------------------------------------
Чтение значений по умолчанию различных переменных.
Аргументы:
ini_name - имя файла или nil (тогда производится последовательный поиск)
section - имя секции
var - имя переменной
default - значение по умолчанию в строковом виде
-------------------------------------------------------------------------|;
```

```
(if ini_name
;| Если задан конкретный файл - читаем из него|;
    (setq result (ru-ini-read ini_name section var default))
;| Если конкретный файл не задан, сначала ищем в СЛОВАРЕ рисунка |;
 (if (not (setq result (ru-dictvar-get-data (strcat section "." var))))
;|Если в словаре рисунка не найдено, читаем из настроечного файла|;
   (setq result
       (ru-ini-read (ru-file-ini "default.ini") section var default))
   );_ end of if
 );_ end of if
  result
);_ end of defun
```

Очень важной функции `ru-dictvar-get-data` и ее окружению мы еще посвятим целый раздел, а пока напишем функцию установки текстового стиля с заданными параметрами (листинг 13.64).

Листинг 13.64. Функция `ru-text-set-style`

```
(defun ru-text-set-style (name file height width oblique / props_list style_obj
style_angle style_file style_height style_name style_width)
;|-------------------------------------------------------------------------
Аргументы:
name - имя стиля
file - имя SHX-файла
height - высота текста
width - ширина
oblique_angle - угол наклона
-------------------------------------------------------------------------|;
  (setq style_obj    (ru-textstyle-obj-add-or-get name)
        props_list   (ru-textstyle-obj-get-props-list style_obj)
        style_file   (nth 0 props_list)
        style_height (nth 1 props_list)
        style_width  (nth 2 props_list)
        style_angle  (nth 3 props_list)
 );_ end of setq
;|-------------------------------------------------------------------------
Вначале при создании файла шрифта может и не быть!
-------------------------------------------------------------------------|;
  (if (= style_file "")
    (setq style_obj
     (ru-textstyle-obj-set-props style_obj file height width oblique)
   );_ end of setq
    (if (not (and
               (= (strcase style_file) (strcase file))
               (= style_height height)
               (= style_width width)
               (= style_angle oblique)
             );_ end of and
      );_ end of not
```

```
      (if (ru-yes
            (strcat "Параметры существующего стиля '" name "'"
               "\n\nФайл шрифта " style_file
               ", высота=" (rtos style_height 2 2)
               ", ширина=" (rtos style_width 2 2)
               ", наклон=" (angtos (ru-conv-deg-to-rad style_angle) 1 4)
               "\n\nне соответствуют заданным:"
               "\n\nФайл шрифта " file
                ", высота=" (rtos height 2 2)
                ", ширина=" (rtos width 2 2)
                ", наклон=" (angtos (ru-conv-deg-to-rad oblique) 1 4)
                 "\n\nИзменить определение стиля"
            );_ end of strcat
          );_ end of ru-yes
         (progn
           (princ (strcat "\nизменяю определение стиля текста '" name
                          "' на заданные параметры"
                  );_ end of strcat
          );_ end of princ
           (setq style_obj (ru-textstyle-obj-set-props
                             style_obj
                             file
                             height
                             width
                             oblique
                           );_ end of ru-textstyle-obj-set-props
          );_ end of setq
           (ru-msg-info
             (strcat
               "Параметры существующего стиля '"
               name
               "'"
               "\n\nизменены на заданные:"
               "\n\nфайл шрифта='" file "', высота="
               (rtos height 2 2) ", ширина=" (rtos width 2 2)
               ", наклон=" (angtos (ru-conv-deg-to-rad oblique) 1 4)
               "\n\nВнимание!"
           "\nначертание существующих примитивов, использующих этот стиль"
               "\nможно переопределить командой"
               "\n'Редакт>Тексты>Приведение стиля к определению'"
             );_ end of strcat
          );_ end of ru-msg-info
        );_ end of progn
      );_ end of if
    );_ end of if
  );_ end of if
   (vla-put-activetextstyle
     (ru-obj-get-active-document)
     style_obj
  );_ end of if
);_ end of defun
```

Функции для работы с объектной моделью AutoCAD

При разработке функций для управления свойствами мы начали обращаться к объектной модели AutoCAD. Давайте сразу разработаем группу функций, облегчающих работу с объектной моделью.

В *главе 10* мы уже достаточно порассуждали о преимуществах и недостатках объектной модели и обещали написать функции для облегчения такой работы. Для демонстрации работы была приведена функция получения списка блоков (см. листинг 10.8), в которой мы последовательно получали указатели на объект `AcadApplication`, в этом объекте получили указатель на `ActiveDocument`, в документе получили указатель на семейство блоков и только затем, проходя по каждому элементу этого семейства, извлекали свойство `Name` и добавляли в традиционный список для языка LISP.

Подобную (в каждом случае свою) цепочку посредников проходить нужно каждый раз при обращению к свойствам и методам объектов. Начинать каждый раз с объекта `AcadApplication`, а заканчивать искомым объектом. Каждый раз придется писать повторяющиеся участки кода и почти наверняка делать в них хотя бы механические ошибки. Иногда надо еще и проверять возможность применения методов к объекту. Если `AcadApplication` существует всегда, то, например, пункт меню может не иметь подпунктов. Такую ситуацию нам уже приходилось учитывать.

Для облегчения работы мы разработаем множество мелких функций. Конечно, еще одна порция "мелочевки" доставит определенные хлопоты, но и результаты скажутся быстро. Разве плохо получить список тех же блоков простым вызовом функции `(ru-obj-list-blocks)`, список слоев — `(ru-obj-list-layers)`, а вообще любой список имен членов семейства — чуть более сложным вызовом `(ru-obj-list-collection-member-names collection)`?

Некоторые постоянно используемые объекты мы будем запоминать в глобальных переменных.

Как получить объект

Начнем с самого главного объекта (листинг 13.65). Обратите внимание, на название листинга "Файл", а не "Функция", как в большинстве случаев. Это вызвано тем, что в файле имеется внефункциональный участок кода, выполняющийся при загрузке. Глобальная переменная `*ru_acad-object*` уничтожается при перегрузке файла. Такой же прием использован и в других случаях (листинги 13.66—13.69).

Листинг 13.65. Файл ru-obj-get-acad-object.lsp

```
(setq *ru_acad-object* nil)
(defun ru-obj-get-acad-object ()
;|-----------------------------------------------------------------------
Возвращает VLA-объект приложения
-----------------------------------------------------------------------|;
  (cond (*ru_acad-object*)
   (t (setq *ru_acad-object* (vlax-get-acad-object)))
 );_ end of cond
);_ end of defun
```

Листинг 13.66. Функция ru-obj-get-active-document

```
(defun ru-obj-get-active-document ()
;|-------------------------------------------------------------------------
Возвращает VLA-объект активного документа
-------------------------------------------------------------------------|;
  (cond (*ru_active-document*)
   (T (setq *ru_active-document*
        (vla-get-activedocument (ru-obj-get-acad-object))
     );_ end of setq
  )
 );_ end of cond
);_ end of defun
```

Листинг 13.67. Функция ru-obj-get-model-space

```
(defun ru-obj-get-model-space ()
;|-------------------------------------------------------------------------
Возвращает VLA-объект пространства модели
-------------------------------------------------------------------------|;
 (cond (*ru_model-space*)
  (t
  (setq *ru_model-space* (vla-get-modelspace
              (ru-obj-get-active-document))))
  );_ end of cond
);_ end of defun
```

Листинг 13.68. Функция ru-obj-get-paper-space

```
(defun ru-obj-get-paper-space ()
;|-------------------------------------------------------------------------
Возвращает VLA-объект пространства листа
-------------------------------------------------------------------------|;
 (cond (*ru_paper-space*)
  (t
  (setq *ru_paper-space* (vla-get-paperspace
            (ru-obj-get-active-document))))
  );_ end of cond
);_ end of defun
```

Листинг 13.69. Функция ru-obj-active-space

```
(defun ru-obj-active-space ()
;|-------------------------------------------------------------------------
Возвращает VLA-объект активного пространства
-------------------------------------------------------------------------|;
 (if (= 1 (vlax-get-property (ru-obj-get-active-document) 'ActiveSpace))
    (ru-obj-get-model-space)
    (ru-obj-get-paper-space)
 )
);_ end of defun
```

А вот пример вполне работающей функции, также возвращающей объект текущего пространства, но написанной так, как привык традиционный LISP-программист (листинг 13.70). Здесь используются и `apply`, и `list`, и все правильно, но преимущество объектного подхода (не знать лишнего о деталях реализации) теряется.

Листинг 13.70. Функция `ru-obj-get-current-space`

```
(defun ru-obj-get-current-space ()
;|-----------------------------------------------------------------------
Пример смеси традиционного и объектного программирования
Так лучше не делать!
-----------------------------------------------------------------------|;
 (apply
   (if (= (getvar "CTAB") "Model")
           'vla-get-modelspace
            'vla-get-paperspace
  );_ end of if
   (list (ru-obj-get-active-document))
);_ end of apply
);_ end of defun
```

Множество операций приходится выполнять с семействами объектов. Не лишними окажутся функции, облегчающие и эту работу (листинги 13.71—13.74).

Листинг 13.71. Функция `ru-obj-acad-collection`

```
(defun ru-obj-acad-collection (name)
;|-----------------------------------------------------------------------
Возвращает VLA-объект семейства с заданным именем
-----------------------------------------------------------------------|;
    (vlax-get-property (ru-obj-get-acad-object) name)
);_ end of defun
```

Листинг 13.72. Функция `ru-obj-collection-count`

```
(defun ru-obj-collection-count (collection)
;|-----------------------------------------------------------------------
Возвращает количество членов семейства
-----------------------------------------------------------------------|;
    (vlax-get-property collection 'Count)
);_ end of defun
```

Листинг 13.73. Функция `ru-obj-collection-list`

```
(defun ru-obj-collection-list (collection / name result)
;|-----------------------------------------------------------------------
Возвращает список членов семейства, переданного в виде объекта
(ru-obj-collection-list (ru-obj-get-layers))
-----------------------------------------------------------------------|;
```

```
(vlax-for each collection
     (setq name (ru-obj-get-obj-name each)
       result (cons name result))
);_ end of vlax-for
    (reverse result)
);_ end of defun
```

Листинг 13.74. Функция ru-obj-doc-collection

```
(defun ru-obj-doc-collection (name)
;|-------------------------------------------------------------------------
Возвращает VLA-объект, заданной по имени семейства, относящегося к активному
документу
-------------------------------------------------------------------------|;
    (vlax-get-property (ru-obj-get-active-document) name)
);_ end of defun
```

Объекты конкретного семейства могут быть получены функциями, подобными приведенным в листингах 13.75—13.77.

Листинг 13.75. Функция ru-obj-get-layers

```
(defun ru-obj-get-layers ()
;|-------------------------------------------------------------------------
Возвращает VLA-объект семейства слоев
-------------------------------------------------------------------------|;
 (ru-obj-doc-collection 'Layers))
```

Листинг 13.76. Функция ru-obj-get-docs-collection

```
(defun ru-obj-get-docs-collection ()
;|-------------------------------------------------------------------------
Возвращает VLA-объект семейства докуметов
-------------------------------------------------------------------------|;
    (ru-obj-acad-collection "Documents")
);_ end of defun
```

Листинг 13.77. Функция ru-obj-docs-count

```
(defun ru-obj-docs-count ()
;|-------------------------------------------------------------------------
Возвращает количество VLA-объектов семейства документов
-------------------------------------------------------------------------|;
    (ru-obj-collection-count (ru-obj-get-docs-collection))
);_ end of defun
```

Функция `ru-obj-docs-list` (листинг 13.78) вернет список открытых документов.

Листинг 13.78. Функция ru-obj-docs-list

```
 (defun ru-obj-docs-list (verbose / doc_name result)
  (vlax-for each (ru-obj-get-docs-collection)
```

```
   (if verbose
     (setq doc_name (strcat (vlax-get-property each 'Path) "\\"
        (ru-obj-get-obj-name each)));_ end of setq
      (setq doc_name (ru-obj-get-obj-name each))
  );_ end of if
   (setq result (cons doc_name result))
 );_ end of vlax-for
  (reverse result)
);_ end of defun
```

Подобным образом можно написать множество других функций для работы с объектной моделью.

Безопасные манипуляции

Мы очень часто напоминаем о необходимости разработки надежных функций, сокращающих возможность возникновения ошибок. Приведем несколько примеров.

Иногда нужно в какую-то функцию передать VLA-объект, хотя нет гарантии, что передаваемый аргумент не является обычным примитивом. В таком случае поможет страховочная функция, показанная в листинге 13.79.

Листинг 13.79. Функция `ru-obj-conv-ename-to-obj`

```
(defun ru-obj-conv-ename-to-obj (entname)
 (cond
  ((= (type entname) 'ENAME) (vlax-ename->vla-object entname))
  ((= (type entname) 'VLA-OBJECT) entname)
);_ end of cond
);_ end of defun
```

При действиях с объектами необходимо предусматривать дополнительные проверки. Посмотрим, как это делается на примере удаления объекта (листинг 13.80).

Листинг 13.80. Функция `ru-obj-delete-object`

```
(defun ru-obj-delete-object (obj)
  (cond
    ((and
       (not (vlax-erased-p obj))
       (vlax-read-enabled-p obj)
       (vlax-write-enabled-p obj)
    );_ end of and
     (vlax-invoke-method obj 'Delete)
     (if (not (vlax-object-released-p obj))
       (vlax-release-object obj)
    );_ end of if
   )
    (T (princ "\nНевозможно удалить объект\n"))
 );_ end of cond
);_ end of defun
```

При получении объекта слоя по имени слоя (листинг 13.81) предусмотрена функция-ловушка ошибки, т. к. при передаче имени несуществующего слоя возникнет ошибка автоматизации. Эта ошибка отлавливается и вместо нее возвращается `nil`.

Листинг 13.81. Функция `ru-obj-layer-by-name`

```
(defun ru-obj-layer-by-name (layer_name / obj_layer)
  (if (vl-catch-all-error-p
    (setq obj_layer (vl-catch-all-apply 'vla-item (list
                          (ru-obj-get-layers) layer_name)
                    );_ end of vl-catch-all-apply
   );_ end of setq
     );_ end of vl-catch-all-error-p
    nil
    obj_layer
 );_ end of if
);_ end of defun
```

Более компактный вариант этой же функции, использующий рассмотренную ранее нашу ловушку ошибок, приведен в листинге 13.82.

Листинг 13.82. Функция `ru-obj-layer-by-name` (вариант)

```
(defun ru-obj-layer-by-name (layer_name)
 (ru-error-catch
    (function (lambda () (vla-item (ru-obj-get-layers) layer_name)))
    (function (lambda (x) nil))
 )
);_ end of defun
```

Пример использования:

```
(ru-obj-layer-by-name  "Несуществующий_слой") вернет nil
(ru-obj-layer-by-name  "0") вернет #<VLA-OBJECT IAcadLayer 011463d4>
```

Как сохранять и восстанавливать собственные данные в файле рисунка[1]

В этом разделе мы, может быть слишком подробно, для того, чтобы хоть раз все-таки показать, *как это делается*, разберем интересную тему о сохранении различных типов данных в файле рисунка.

Разработчику, работающему в любой системе, при написании даже очень простых программ, нередко требуется что-то запомнить для последующего использования, а в сложных программах это происходит практически постоянно. AutoLISP, в данном случае, совсем не исключение. Но прежде чем говорить о хранении данных, необхо-

[1] Раздел написан Петром Лоскутовым.

димо определить, что мы собираемся хранить, когда оно нам понадобится, и какие механизмы мы будем для этого использовать. Кроме того, нужно четко понимать, каким образом в AutoCAD организована "видимость" данных. Чаще всего "хранилищем" данных в AutoLISP являются переменные, но это не единственный способ сохранения информации.

Для начала, кратко напомним самые простые моменты — с "видимостью" переменных в AutoLISP, или иначе — с "областью их определения". По этому признаку переменные обычно делят на две категории: *глобальные* и *локальные* переменные. Такое деление не совсем точно, но поскольку оно является общепринятым, мы будем его придерживаться. Упомянем только, что кроме "обычных" глобальных переменных существуют еще и "междокументные" переменные, нечто вроде "суперглобальных" переменных, областью видимости которых условно можно считать уровень приложения. Поскольку этот раздел в настоящий момент нас не интересует, далее под *глобальными* мы будем понимать переменные уровня документа.

Как уже сказано, областью определения глобальных переменных является документ, т. е. любая функция, выполняемая в контексте рисунка, может получить доступ к этим переменным, а значит, как получить, так и изменить их значение. Вполне очевидно, что именно с этим свойством связаны как положительные, так и отрицательные качества глобальных переменных. Используя глобальные переменные, нельзя быть *абсолютно* уверенным, что значение, оставленное в глобальной переменной "без присмотра", не было изменено другой программой в результате простого совпадения имен. Чаще всего такие совпадения бывают либо при использовании разных программ одного разработчика, либо при использовании функций других разработчиков. Единственным способом хоть как-то защитить глобальную переменную от изменения сторонней программой является уникальность ее имени. У нас глобальные переменные имеют вид `*ru_имя_переменной*`. Гарантий это никаких не дает, но риск несколько снижает.

Локальные переменные определяются в функциях их создающих, а это означает, что вне порождающих функций такие переменные не существуют. Ограничение области видимости переменных дает им очень полезное свойство — можно не беспокоиться о согласовании имен переменных, чтобы ненароком не нарушить работу программ других разработчиков, не нужно выдумывать все более сложные имена переменным и вести их реестр, не нужно заботиться о "приборке" по завершении работы функции. Одним словом, использование локальных переменных *безопасно*. Вроде бы все замечательно, но хранить данные в локальных переменных можно только пока выполняется функция, в которой они определены. При завершении функции все локальные переменные уничтожаются. Даже при одновременном выполнении нескольких экземпляров одной функции значения одноименных переменных в них может быть разным, поскольку это разные переменные.

Для того чтобы закончить тему области определения переменных, разберем одно свойство *видимости* переменных, которое бывает неочевидным для начинающих разработчиков[1]. Одна и та же переменная может быть локальной для одной функции и "глобальной" для другой. Проявляется это в том, что если в функции `A` определена локальная переменная, и эта функция вызывает функцию `B`, то локальная перемен-

[1] Эффект возникает, когда разработчик использует одинаковые имена для глобальных и локальных переменных.

ная функции A будет "глобальной" для функции B. Но "глобальность" эта мнимая. Проиллюстрируем это свойство простейшим примером — создадим несколько демонстрационных функций и посмотрим, как они будут работать (листинг 13.83).

Листинг 13.83. Демонстрация видимости переменных

```
(defun var-test-loc (/ myvar)
  (setq myvar "Это локальная переменная.")
  (princ myvar)
  (princ)
);_ end of defun
(defun var-test-glob ()
  (princ myvar)
  (princ)
);_ end of defun
(defun var-test-sub1 (/ myvar)
  (setq myvar "Это локальная переменная.")
  (var-test-glob)
);_ end of defun
(defun var-test-sub2 (/ myvar)
  (setq myvar "Это локальная переменная.")
  (var-test-sub-sub)
);_ end of defun
(defun var-test-sub-sub ()
  (var-test-glob)
);_ end of defun
```

Загрузим функции в систему AutoCAD и приступим к манипуляциям. Первым делом создадим настоящую глобальную переменную с таким же именем, как и в загруженных функциях:

```
_$ (setq myvar "Это глобальная переменная.")
"Это глобальная переменная."
```

Убедимся, что простое обращение к глобальной переменной приводит к ожидаемому результату:

```
_$ (var-test-glob)

Это глобальная переменная.
```

Теперь посмотрим, какое значение получит AutoLISP, обратившись к переменной с тем же именем, но внутри функции, имеющей одноименную локальную переменную:

```
_$ (var-test-sub1)
Это локальная переменная.
```

"Усугубим" ситуацию, обратившись к той же переменной не сразу в вызываемой функции, а в функции, которую вызывает функция, вызванная первой:

```
_$ (var-test-sub2)
Это локальная переменная.
```

Ну и напоследок проверим состояние глобальной переменной:

```
_$ (var-test-glob)
Это глобальная переменная.
```

Как видно на этом примере, локальная переменная, определенная в вызывающей функции, становится "глобальной" по отношению *ко всем* вызываемым функциям, если эти функции не имеют одноименных локальных переменных.

Совершенно очевидно, что для хранения данных в промежутках между вызовами функций локальные переменные совершенно непригодны. С глобальными переменными ситуация не столь очевидная и требует некоторых пояснений. Особенно с учетом того обстоятельства, что именно они часто используются для хранения данных, хотя многие понимают, что это "не всегда хорошо". Если данные требуется сохранить в рисунке между сеансами редактирования, все усложняется еще больше. В AutoCAD предусмотрено 15 пользовательских системных переменных, с "древних" времен предназначавшихся для таких целей. Но их количество невелико (можно сохранить по пять целых, вещественных чисел и строк), к тому же программы разных разработчиков могут использовать эти переменные по собственному разумению, т. е. возможны конфликты, аналогичные проблеме совпадения имен при использовании глобальных переменных, но с гораздо большей вероятностью. Сложившаяся в последнее время практика бесконтрольного использования этих переменных делает их совершенно непригодными для целей длительного хранения данных.

В ранних версиях системы AutoCAD мы сохраняли все необходимые переменные в INI-файлах, часто делаем это и сейчас. Некоторые данные, требующиеся для использования в иных программах (например, в системе документооборота), требуется хранить во внешних файлах, но многое можно и нужно хранить в самом рисунке. AutoCAD теперь имеет много укромных местечек в рисунке, куда спрятать можно все, что требуется. Одним из таких мест являются *словари* (Dictionaries). Система AutoCAD хранит в них множество данных, и, что важнее, благородно предоставляет пользователю такие же возможности. Воспользуемся этим и создадим набор функций, предназначенных для создания и хранения в файле чертежа данных, доступ к которым должен осуществляться примерно так же, как при использовании глобальных переменных, но в том числе и в разных сессиях редактирования чертежа.

Прежде чем начинать разработку собственного механизма, посмотрим, что предлагает для этих целей Visual LISP из встроенных функций. В данном случае интерес для нас представляют функции с префиксом `vlax-ldata-`. Во многих случаях возможностей, которые предлагают эти функции, вполне достаточно и необходимости в разработке собственных функций не возникает. Мы не будем подробно разбирать синтаксис встроенных функций, ограничимся кратким перечислением их возможностей, потенциальными проблемами, которые могут возникнуть при их применении, и естественно, самое пристальное внимание обратим на пути решения этих проблем. Функции этой группы позволяют производить следующие операции:

- `vlax-ldata-put` — сохраняет данные в указанном словаре;
- `vlax-ldata-get` — читает данные, сохраненные в словаре;
- `vlax-ldata-delete` — удаляет запись из словаря;
- `vlax-ldata-list` — возвращает список записей словаря;
- `vlax-ldata-test` — проверяет данные на допустимость хранения их в словаре.

Получается, что все необходимые действия вроде бы уже реализованы и, возможно, никакой необходимости в рассмотрении данной темы именно в этой книге нет. Функции эти действительно очень полезные и удобные, но есть пара нюансов, которые не позволяют обойтись, особенно в некоторых ситуациях, исключительно ими.

Во-первых, эти функции не позволяют хранить в словарях наборы примитивов. Причем, что особенно неприятно, сохранить в словаре набор можно, но при попытке получить его обратно или, что уже совсем отвратительно, сохранить такой файл, происходит ошибка. Такой ошибкой можно поставить пользователя в совершеннейший тупик — шансы на то, что он догадается вручную почистить словарь, практически равны нулю.

Примечание

Для формирования и хранения в рисунке объединений примитивов в системе AutoCAD также предусмотрено другое средство — *группы*, которые является аналогами наборов. Группы создаются командой GROUP (ГРУППА) и хранятся в словаре с зарезервированным именем ACAD_GROUP. *(Николай Полещук)*

Во-вторых, при сохранении в словаре имен примитивов, которые впоследствии были удалены, иногда также происходит ошибка, как при сохранении документа, так и при попытке доступа к записи, содержащей имя несуществующего примитива.

Поскольку хранение имен примитивов[1], а также наборов примитивов представляет особый интерес для разработчиков хитрых программ, эти особенности встроенных функций могут доставлять изрядные неприятности. На резонное в других условиях замечание, что за целостностью данных должен следить разработчик программы, можно ответить, что речь идет, в основном, о хранении данных *вне* контекста выполнения программы, которая ими пользуется. Соответственно, возможность контролировать содержимое записей существенно снижается.

Замечание

Не следует воспринимать хранение данных в словаре, как альтернативу глобальным переменным. Помимо наличия очень похожей, хотя и не такой частой, проблемы согласования имен, данный механизм работает много медленнее, чем механизм обращения к переменным. Хранение в словаре актуально в первую очередь для тех данных, которые должны быть сохранены между сеансами редактирования документа.

Итак, проблема у нас сформулирована, попробуем сделать корректную постановку задачи. Нам необходим механизм, позволяющий хранить в словаре как имена примитивов, так и наборы примитивов. Хорошо бы, чтобы он позволял хранить и другие типы данных, в том числе сложные списки, в едином месте и с единым способом доступа. Этот механизм не должен быть чувствителен к удалению примитивов, хранящихся как в виде имен, так и виде наборов. При хранении набора нас интересует его содержимое, а не набор, как объект. Доступ к данным должен быть как можно более похожим на использование переменных, т. е. предельно облегченным.

---

[1] Строго говоря, имена примитивов постоянны только в текущем сеансе редактирования. Неизменными сохраняются метки примитивов, по которым с помощью функции `handent` можно получать действующие имена примитивов.

Конечно, столь упрощенная постановка задачи допустима только при разработке небольшого набора функций, выполняющих локальную и достаточно очевидную задачу. Тем не менее даже такая формализация задачи позволяет лучше подготовиться к ее решению.

Теперь настал черед определения набора интерфейсных функций для решения поставленных задач. Опишем их на уровне выполняемых действий, форматов вызова, аргументов и возвращаемых значений. Пока нас не интересует ни их внутренняя структура, ни, тем более, детали реализации. Гораздо важнее сразу предусмотреть такой интерфейс, чтобы его не пришлось переделывать. Необходимость в переделке интерфейса может возникнуть как из-за забытого, но нужного аргумента, так и из-за излишне усложненного интерфейса, перегруженного ненужными аргументами.

Как минимум, нам понадобятся две базовые функции:

- `(ru-dictvar-set-data <var_name> <value>)` — для сохранения данных в словаре в именованной записи (далее в данном разделе будем называть такую запись *переменной*), где:
  - `<var_name>` — строка, имя переменной;
  - `<value>` — значение переменной.
- `(ru-dictvar-get-data <var_name>)` — для доступа к сохраненным данным, где:
  - `<var_name>` — строка, имя переменной.

Кроме того, было бы очень неплохо иметь несколько сервисных функций для упрощения некоторых весьма вероятных действий:

- `(ru-dictvar-clear-data <var_name>)` — для уничтожения переменной, где:
  - `<var_name>` — строка, имя переменной.
- `(ru-dictvar-clear-all)` — для уничтожения словаря и всех переменных.
- `(ru-dictvar-list-all-var)` — для получения списка имен всех имеющихся в словаре переменных.

Для удовлетворения собственного самолюбия и для некоторой пользы предусмотрим еще пару функций:

- `(ru-dictvar-copyright-data)` — для получения краткой информации о словаре и реквизитов разработчика в специальном формате.
- `(ru-dictvar-about-dlg)` — для того же, но в виде информационного окна.

Остановимся немного подробнее на аргументах основных функций. Среди этих аргументов отсутствует какое-либо упоминание имени словаря, а как мы знаем, словарь является именованным объектом и при его создании имя указывать придется. С точки зрения универсальности было бы правильнее передавать имя словаря при вызове наших функций, по аналогии со встроенными функциями. Но, во-первых — это увеличило бы вероятность ошибок при передаче аргументов, а во-вторых — плодить пользовательские словари, в каждом из которых, скорее всего, будет по паре записей, также не лучший путь. Поэтому было принято, возможно, спорное решение зашить имя словаря внутрь механизма, правда, не очень глубоко. То есть мы как бы резервируем за собой словарь с конкретным именем и набор функций для работы с ним, которые имеют строго фиксированные имена. Вероятно, впоследствии и

будет принято решение об извлечении имени словаря из функций и организации передачи имени через дополнительный аргумент, но в этом случае все затронутые этими изменениями функции будут обязательно переименованы — соответственно, это будут *новые* функции.

Еще раз обратим взгляд на базовые функции. Единственное их назначение — организация удобного интерфейса между внешними функциями, из которых производится обращение к разрабатываемому нами механизму и внутренними служебными функциями, предназначенными для непосредственных манипуляций со словарями AutoCAD и другими объектами. Фактически сейчас нам надо только определить типы данных, которые будет принимать функция сохранения данных, и какие действия должны производить служебные функции, которые будут вызываться интерфейсными. Начнем с описания типов данных, которые функция сохранения должна принимать, включая описание особенностей и ограничений хранения данных:

- INT — целые числа;
- REAL — числа с плавающей точкой;
- STR — строки;
- ENAME — имена примитивов, сохраняющие актуальность[1] между сеансами редактирования;
- VLA-OBJECT — VLA-объекты, сформированные по именам некоторых примитивов (ENAME) и восстанавливающиеся при возврате;
- PICKSET — наборы объектов, хранящиеся в виде списков объектов типа ENAME; при возврате создается новый набор, содержащий те же примитивы; примитивы, окончательно удаленные из графической базы данных, в набор не включаются;
- LIST — списки, в том числе точечные пары, содержащие только поддерживаемые типы данных, точечные пары стандартного вида: `(<INT> . <данные>)`, поддерживаются вложенные списки.

Функция возвращает *фактически сохраненное значение*. В случае отсутствия словаря, создает его.

С какой целью нам понадобилось включать в поддерживаемые типы числа, строки, списки, т. е. те типы данных, которые можно поместить в словарь при помощи встроенных функций? В первую очередь, раз уж мы беремся за создание механизма хранения данных, то должны предусмотреть хранение как можно большее широкого набора типов для того, чтобы наш механизм был универсальным. Это существенно упростит его использование за счет единообразия решений в тех ситуациях, когда необходимость в нем возникнет. У того, кто захочет воспользоваться плодами нашей работы, не должно быть проблем, связанных с ограниченностью возможностей использования тех или иных данных. Вторая причина заключается в том, что LISP, будучи языком списков, подразумевает использование данных также в виде списков, и если мы не предусмотрим хранение списков, то наш механизм потеряет значительную часть своей привлекательности. В списках же неизбежно будут встречаться простые типы данных. Кроме того, еще на стадии постановки задачи можно обос-

[1] Актуальность не означает неизменность.

нованно предположить, что хранение простых типов данных организовать будет гораздо легче, чем сложных.

Поскольку исходные тексты функций, рассматриваемых в этом разделе, можно найти на прилагаемом компакт-диске в полном объеме, здесь мы подробно рассмотрим, для примера, только одну функцию. В основном же сосредоточимся на принципах и приемах использованных при разработке механизма хранения данных в словаре.

Посмотрим, какие действия должны происходить в функции сохранения данных `ru-dictvar-set-data`, на уровне схемы:

1. Найти словарь, в котором будут храниться данные. В случае отсутствия словаря, создать его. Привязать словарь к переменной.
2. Проверить наличие в словаре записи с именем, которое передано в качестве аргумента, и при наличии такой записи удалить[1] ее.
3. Если значение аргумента `<value>`, переданное в функцию, не `nil`, поместить в словарь новую запись с соответствующими параметрами.
4. Вызвать функцию получения данных из словаря и вернуть возвращенное ей значение при выходе.

Для столь простой функции рисовать схемы и разбираться с отдельными блоками не принято, а при наличии некоторого опыта большую часть этой работы можно проводить в уме. Но, имея цель отработать методику и продемонстрировать принципы последовательной разработки, гораздо нагляднее показать все на простом примере, проведя весь цикл, пусть даже и в избыточном варианте. Итак, разберем содержимое, которое должно быть в каждом блоке, на уровне описания.

Примечание

Вынос описания блоков из схемы в отдельный блок для простых функций также не характерен, как подробное их описание, но в случае разработки сложных функций, особенно в составе команды, наличие упрощенной схемы позволяет, при необходимости, создавать схемы программ, состоящих из схем функций.

Блок № 1. Вызвать функцию `_ru-dictvar-get-dict` получения DXF-списка словаря и, если список получен, извлечь из него имя примитива словаря. В случае если вместо списка возвращен `nil`, вызвать функцию `_ru-dictvar-add-new-dict` создания нового словаря, которая должна вернуть имя примитива словаря. Результат поместить в локальную переменную.

Блок № 2. Провести поиск по словарю записи с указанным именем, если найдена, вызвать функцию `_ru-dictvar-remove-var`, которая удалит эту запись.

Блок № 3. Проверить переданный аргумент на "содержательность", и при положительном результате вызвать функцию `_ru-dictvar-set-var-data` создания новой записи с указанными именем и значением.

Блок № 4. Вызвать функцию `ru-dictvar-get-data`, которая вернет фактически сохраненное значение. Поскольку после вызова функции `ru-dictvar-get-data` никаких действий больше не происходит, то ее возвращаемое значение будет возвращаемым значением и для функции `ru-dictvar-set-data`.

---

[1] X-записи перед заменой необходимо удалять.

После того как мы разобрались со всем этим, написание собственно кода функции превращается в весьма рутинную, механическую операцию. Это достаточно наглядно демонстрирует листинг 13.84. Функции, имена которым мы выбрали при создании описания, предстоит написать позднее. На данном этапе нам достаточно знать, как их вызвать, т. е. какие они имеют аргументы, и что возвращают. Вполне возможно, что какие-то из них будут разрабатываться другими людьми.

Листинг 13.84. Функция `ru-dictvar-set-data`

```
(defun ru-dictvar-set-data (var_name value / user_dict tmp)
;;; Начало блока №1. ------------------------------------------
    (setq user_dict (if (not (setq tmp (_ru-dictvar-get-dict)))
                        (_ru-dictvar-add-new-dict (namedobjdict))
                        (cdr (assoc -1 tmp))
                    );_ end of if
   );_ end of setq
;;; Конец блока № 1. ------------------------------------------
;;; Начало блока № 2. ------------------------------------------
    (if (dictsearch user_dict var_name)
        (_ru-dictvar-remove-var user_dict var_name)
   );_ end of if
;;; Конец блока № 2. ------------------------------------------
;;; Начало блока № 3. ------------------------------------------
    (if value
        (_ru-dictvar-set-var-data
            var_name
            (_ru-dictvar-add-new-var-name user_dict var_name)
            value
       );_ end of _ru-dictvar-set-var-data
   );_ end of if
;;; Конец  блока № 3. ------------------------------------------
;;; Начало блока № 4. ------------------------------------------
    (ru-dictvar-get-data var_name)
;;; Конец  блока № 4. ------------------------------------------
);_ end of defun
```

Здесь приведен уже окончательный листинг функции. Таким он становится, естественно, не сразу. В первых набросках вызовы еще неопределенных функций могут показываться условно, а к окончательному виду они приводятся после проработки задания на вызываемые функции. При этом могут возникать новые функции, необходимость в которых выявляется в ходе работы.

Поскольку остальные интерфейсные функции разработать ничуть не сложнее, чем только что приведенную, а их листинги не дадут ничего нового в плане показа принципов или приемов работы, то мы и не будем демонстрировать их в книге. Интереснее рассмотреть некоторые принципы операций с данными, которые мы будем хранить в словаре.

Когда мы определяли то, какие типы данных будем хранить в словаре, ни слова не было сказано о способе хранения. Для нашего случая достаточно знать, что в словарях данные можно хранить в виде X-записей (XRECORD). Каждая X-запись представляет собой DXF-список, выглядит который примерно так:

```
(
;;; Начало "служебного блока" --------------------
    (-1 . <Entity name: XXXXXXXX>)
    (0 . "XRECORD")
    (5 . "1E07")
    (102 . "{ACAD_REACTORS")
    (330 . <Entity name: XXXXXXXX>)
    (102 . "}")
    (330 . <Entity name: XXXXXXXX>)
    (100 . "AcDbXrecord")
    (280 . 1)
;;; Конец "служебного блока" ---------------------
;;; Начало "блока пользовательских данных" ------
    (<код> . <данные>)
    (<код> <число> <число> [<число>])
    ...
;;; Конец "блока пользовательских данных" -------
)
```

Служебный блок нас пока не интересует, а вот в пользовательском блоке мы как раз собираемся хранить свои данные. Фактически наш механизм должен сформировать корректный DXF-список пользовательского блока, приставить к нему спереди необходимые служебные записи и создать новый примитив. Данные в DXF-списке хранятся либо в виде точечных пар, либо в виде плоских списков определенного формата. Туда нельзя поместить ни произвольный плоский список, ни сложный список, т. е. имеющий вложенные списки. Это не помешает хранить простые данные, достаточно определить, какие коды мы будем использовать для создания точечных пар[1], разработать простейший механизм формирования таких пар и механизм для формирования DXF-списка будущей записи. А вот для того чтобы иметь возможность хранения в словаре списков произвольного формата, нам придется разработать собственную систему прямого и обратного преобразования списков. Прямое преобразование должно превращать любой список в список точечных пар и списков, пригодных для помещения в DXF, а обратное преобразование должно восстанавливать из такого списка исходный[2]. Для решения этой задачи нам придется ввести дополнительные точечные пары, которые будут означать начало и конец списка. Сделать это можно следующим образом. Пусть у нас есть сложный список:

```
(
    1234
    "некий текст"
    (
        "текст в подсписке"
        5678.9
   )
```

[1] Какие коды использовать для хранения разных типов данных, зависит от разработчика. Выбор можно сделать по информации, содержащейся в справке AutoCAD.

[2] Преобразование списка в строку с последующим прочтением строки при восстановлении, как это реализовано во встроенных функциях, не подходит, по причине усложнения интерпретации при хранении наборов и удаленных примитивов.

```
    "еще немного текста"
    (12.3 45.6 78.9)
    0
    (100 . "текст в точечной паре")
)
```

Список из трех чисел не разбит по причине того, что такой список удобно хранить без переформатирования, как координаты точки, а для обозначения точечных пар мы предусмотрим специальный код. Конвертируем исходный список при помощи функции `_ru-dictvar-gen-var-list` в список элементов, корректных для DXF:

```
(
    (400 . 1234)
    (300 . "некий текст")
    (102 . "{")
    (300 . "текст в подсписке")
    (40 . 5678.9)
    (102 . "}")
    (300 . "еще немного текста")
    (10 12.3 45.6 78.9)
    (400 . 0)
    (401 . 100)
    (300 . "текст в точечной паре")
)
```

Хорошо видна прямая связь точечных пар, имеющих код 102, с началом и концом вложенного списка. Нами определено, что пары с этим кодом могут, в частности, содержать в качестве данных два значения: `"{"` или `"}"`, что позволит при обратном преобразовании определить, где начинается список и где он заканчивается. Простые элементы списка и подсписка преобразованы в точечные пары с соответствующими кодами. Точечная пара исходного списка превратилась в две пары, у первой из которых код 401, в отличие от кода 400, применяемого для целых чисел в других случаях. При обратном преобразовании это даст знать, что целое число, хранящееся в данной точечной паре в качестве данных, является первым элементом (кодом) сохраненной точечной пары, а следующий элемент DXF-списка — это второй элемент (данные) той же пары.

Разобравшись со способом хранения данных простых типов, точечных пар и списков, обратим внимание на данные, ради хранения которых и был затеян весь этот "геморрой". Имена примитивов мы будем хранить так же, как это делают встроенные функции, с той лишь разницей, что при возвращении отдельного примитива не будем проверять его на удаленность. Решение, возможно, и не бесспорное, но таким образом мы сохраняем возможность отслеживания того, был ли удален искомый примитив или его не существовало никогда. Такое отслеживание дает возможность "обругать" пользователя более содержательно.

Формат DXF-списков не позволяет хранить имена примитивов, представленные в виде VLA-объектов, поэтому для их хранения также применим методику, аналогичную встроенным функциям — будем преобразовывать их в обычные примитивы AutoCAD и хранить в словаре в таком виде. Для того чтобы при возвращении примитива знать, что его нужно вернуть в виде VLA-объекта, поместим пару со специальным кодом перед парой, хранящей примитив.

В DXF-списке это будет выглядеть следующим образом:

```
(102 . "{ru_Global_Variable")
(402 . 1)
(-2 . <Entity name: 402b3678>)
(102 . "ru_Global_Variable}")
```

Здесь показан весь *пользовательский блок*, включая начальный и конечный элементы с кодом 102. Наличие таких элементов упрощает определение границ обработки при извлечении данных. Имя примитива сохранено в паре с кодом −2, так же как и в случае передачи на сохранение имени примитива, но перед этой парой помещена пара с кодом 402 и значением 1. Наличие такой пары сообщает функции, ответственной за извлечение данных, что следующая пара является именем примитива, который нужно преобразовать перед возвращением.

Осталось придумать способ корректного хранения *наборов примитивов* (PICKSET). Сохранить набор примитивов в DXF-списке нельзя, но можно сохранить список имен примитивов. Тогда мы можем преобразовать набор в список примитивов, сохранить этот список, а при передаче сохраненного значения, содержащего набор, восстановить его. Сделать это можно путем создания нового набора и включения в него примитивов первоначального набора. Естественно, восстановленный набор не будет тем же самым набором, который был передан на хранение, но он будет содержать те же примитивы, что и исходный. Этого вполне достаточно для того, чтобы считать восстановленный набор эквивалентным исходному набору, за исключением только того, что в возвращаемый набор нельзя включать удаленные примитивы. Так что при восстановлении набора необходимо предусмотреть проверку примитива на его существование.

Почти все, что требуется для решения задачи хранения наборов в словаре, у нас уже есть. Мы предусмотрели хранение списков и примитивов. Осталось написать или использовать уже имеющиеся функции преобразования набора в список и наоборот, ну и определить, каким способом мы будем маркировать такие списки, чтобы при возвращении знать о необходимости вернуть набор, а не список.

Функция преобразования набора в список есть в арсенале каждого разработчика (они бывают всего трех типов), поэтому на ней останавливаться не будем. Обратное преобразование также не представляет никаких сложностей, но встречается несколько реже. Кроме того, в нем есть уже обозначенная особенность — необходимость проверки примитивов перед помещением в набор. Для этой цели разработана простая рекурсивная функция `_ru-dictvar-list-to-pickset` (листинг 13.85).

Листинг 13.85. Функция _ru-dictvar-list-to-pickset

```
(defun _ru-dictvar-list-to-pickset (lst / tmp)
    (if (setq tmp (car lst))
        (if (entget tmp)
            (ssadd tmp (_ru-dictvar-list-to-pickset (cdr lst)))
            (_ru-dictvar-list-to-pickset (cdr lst))
       );_ end of if
        (ssadd)
   );_ end of if
);_ end of defun
```

Посмотрим, как будет выглядеть пользовательский блок записи, содержащей набор из трех примитивов:

```
(102 . "{ru_Global_Variable")
(402 . 2)
(102 . "{")
(-2 . <Entity name: 402b36a0>)
(-2 . <Entity name: 402b3698>)
(-2 . <Entity name: 402b3678>)
(102 . "}")
(102 . "ru_Global_Variable}")
```

Внимания тут заслуживает пара с кодом 402. Если, в случае с хранением VLA-объекта, пара с аналогичным кодом имела значение 1, то здесь значение равно 2. Теперь становится ясно, что код 402 означает необходимость дополнительной интерпретации хранимых данных, а значение такой пары указывает на то, что должно следовать за этой парой и в каком формате данные должны быть возвращены. Значение 2, в частности, указывает, что список, который следует дальше (а то, что это список, ясно благодаря парам с кодом 102), должен быть преобразован в набор примитивов перед возвращением.

Для того чтобы убедиться в работоспособности механизма, посмотрим, как может выглядеть полный DXF-список записи, хранящей сложные данные. Попробуем сохранить в качестве данных вот такой список:

```
(
    123
    45.6
    "текст"
    (78.0 9.1 2.3)
    <Entity name: 402b36a0>
    (
        "текст в подсписке"
        4
        5.0
   )
    <Selection set: f>
    (
        98.0
        (
            <Entity name: 402b3678>
            (21926.0 16609.0 0.0)
       )
   )
)
```

А теперь посмотрим содержимое DXF-списка записи словаря в виде таблицы с разбором значений каждой точечной пары (табл. 13.1).

Обращение к записи возвращает список, эквивалентный исходному списку. Простой анализ позволяет установить однозначное соответствие элементов исходного списка и данных, содержащихся в записи. Таким образом можно считать, что с поставленной задачей мы справились.

Таблица 13.1. *Содержимое списка записи словаря*

Элемент	Пояснение
(-1 . <Entity name: 402b36b0>)	Имя примитива X-записи
(0 . "XRECORD")	Тип примитива
(5 . "1E1E")	Метка примитива X-записи
(102 . "{ACAD_REACTORS")	Метка начала данных связи с реактором
(330 . <Entity name: 402b3620>)	Имя примитива словаря, владельца записи
(102 . "}")	Метка конца данных связи с реактором
(330 . <Entity name: 402b3620>)	Имя примитива словаря
(100 . "AcDbXrecord")	Тип примитива
(280 . 1)	Флаг управления дублированием записи
(1 . "Test for book")	Имя записи
(102 . "{ru_Global_Variable")	Начало данных, хранимых в записи
(102 . "{")	Начало списка
(400 . 123)	Целое число
(40 . 45.6)	Число с плавающей точкой
(300 . "текст")	Текст
(10 78.0 9.1 2.3)	Список, соответствующий формату координат точки
(-2 . <Entity name: 402b36a0>)	Примитив AutoCAD
(102 . "{")	Начало списка
(300 . "текст в подсписке")	Текст
(400 . 4)	Целое число
(40 . 5.0)	Число с плавающей точкой
(102 . "}")	Конец списка
(402 . 2)	Признак, что следующий список необходимо трактовать как набор
(102 . "{")	Начало списка
(-2 . <Entity name: 402b36a0>)	Примитив AutoCAD в наборе
(-2 . <Entity name: 402b3698>)	Примитив AutoCAD в наборе
(102 . "}")	Конец списка
(102 . "{")	Начало списка
(40 . 98.0)	Число с плавающей точкой
(102 . "{")	Начало списка
(-2 . <Entity name: 402b3678>)	Примитив AutoCAD в списке
(10 21926.0 16609.0 0.0)	Список, соответствующий формату координат точки

Таблица 13.1 *(окончание)*

Элемент	Пояснение
(102 . "}")	Конец списка
(102 . "}")	Конец списка
(102 . "}")	Конец списка
(102 . "ru_Global_Variable}")	Конец данных, хранимых в записи

Безусловно, это не единственный из возможных способов решения задачи долговременного хранения данных, но в рамках этой разработки, помимо решения основной задачи, мы приобрели опыт "закапывания" своих данных в документе минуя стандартные механизмы, что может пригодиться и в других ситуациях. С минимальными переделками разработанный механизм можно приспособить для сохранения данных не в словарях, а в расширенных данных примитивов или в другом укромном уголке. Для разработчика такая возможность может оказаться весьма полезной во многих случаях. AutoCAD предоставляет некоторые готовые механизмы для подобных целей, но научиться произвольно оперировать записями при помощи этих механизмов сложно. Такая манипуляция может существенно расширить возможности, доступные разработчику, который найдет время для того, чтобы самостоятельно пройти весь цикл создания подобного механизма.

Работа с меню

В числе первоочередных нам потребуются несколько функций для загрузки своего меню и управления его разделами. Так как собственное меню во время разработки нам придется перезагружать сотни раз, напишем для этого специальную функцию (листинг 13.86). Перегружать меню мы будем из MNU-файла, оригинал которого будет "проживать" в подкаталоге %RootDir%\All Users\template\. Мы поддерживаем, как минимум, две версии системы AutoCAD, а редактировать лучше единственный файл. При перегрузке исходный файл меню будет просто копироваться в каталог файлов поддержки соответствующей версии AutoCAD.

Листинг 13.86. Функция `ru-menu-reload`

```
(defun ru-menu-reload ()
;|-------------------------------------------------------------------------
 Перегрузка и перекомпилирование главного меню
-------------------------------------------------------------------------|;
  (vl-file-delete (strcat (ru-dirs-acad nil) "ruCAD.mnc"))
  (vl-file-delete (strcat (ru-dirs-acad nil) "ruCAD.mnr"))
  (vl-file-delete (strcat (ru-dirs-acad nil) "ruCAD.mns"))
  (vl-file-delete (strcat (ru-dirs-acad nil) "ruCAD.mnu"))
  (vl-file-copy (ru-file-template "ruCAD.mnu")
        (strcat (ru-dirs-acad nil) "ruCAD.mnu"))
 (ru-menu-ru-load)
 (princ "\nМеню перегружено и перекомпилировано\n")
 (princ)
);_ end of defun
```

Простая функция (листинг 13.87) загрузки нашего меню без перекомпилирования. Обычный пользователь должен иметь возможность быстро переключаться между меню ruCAD и меню стандартной системы AutoCAD.

Листинг 13.87. Функция `ru-menu-ru-load`

```
(defun ru-menu-ru-load ()
;|-------------------------------------------------------------------------
Загрузка файла меню ruCAD
-------------------------------------------------------------------------|;
 (ru-obj-set-preferences-key
    "Files"
    "MenuFile"
    (strcat (ru-dirs-acad nil) "ruCAD")
 );_ end of ru-obj-set-preferences-key
;;; Важнейшая функция установки доступа к пунктам меню
  (ru-menu-set-enabled T)
  (princ)
);_ end of defun
```

Соответственно должна быть функция загрузки стандартного меню системы AutoCAD (листинг 13.88). В стандартном меню нет пункта для возврата в наше меню, поэтому предусматриваем переопределение команды RU (листинг 13.89), с помощью которой можно вернуться в меню ruCAD. Вообще-то мы даже можем предусмотреть специальную кнопку в статусной строке, но сделаем это после знакомства с COM-технологиями.

Листинг 13.88. Функция `ru-menu-acad-load`

```
(defun ru-menu-acad-load ( / file_name)
;|-------------------------------------------------------------------------
Загрузка базового меню AutoCAD
-------------------------------------------------------------------------|;
 (ru-obj-set-preferences-key "Files" "MenuFile"
    (strcat (ru-dirs-acad nil) "acad")
 );_ end of ru-obj-set-preferences-key
(princ (strcat "\nЗагружено меню " file_name
               "\nДля возврата в ruCAD набери 'RU'\n"
       );_ end of strcat
 )
 (princ)
)
```

Листинг 13.89. Фрагмент файла ruCAD.mnu

```
ID_MnTools     [&Настройки]
RU_ACAD_MENU   [~Меню  Acad]^C^C^P(defun C:RU()+
                        (ru-menu-ru-load)(princ));(ru-menu-acad-load)
```

Вывод специальных меню

В *главе 8* мы писали о том, что будем подменять в разделе POP6 падающее меню специальных рисунков на меню рабочей группы пользователя. Реализуем эту идею (листинги 13.90 и 13.91).

Листинг 13.90. Функция ru-menu-special-show

```
(defun ru-menu-special-show (name)
;|-------------------------------------------------------------------------
Замена в разделе ***POP6 меню специальных рисунков на меню рабочей группы текущего
пользователя
-------------------------------------------------------------------------|;
 (menucmd (strcat "P6=" name))(menucmd "P6=*")
(princ)
);_ end of defun
```

Листинг 13.91. Функция ru-menu-special-common-show

```
(defun ru-menu-special-common-show ()
;|-------------------------------------------------------------------------
 Вывод оглавления всех специальных меню с отметкой меню рабочей группы
-------------------------------------------------------------------------|;
 (ru-menu-tag-check-rucad-workgroup)
;; Именно так - сначала отметить, потом показать
(ru-menu-special-show "common")
(princ)
);_ end of defun
```

Загрузка и выгрузка фрагментных меню

Далее пишем несколько функций (листинги 13.92—13.94) для загрузки *фрагментных меню*. Вообще-то наше меню не будет загружаться, как фрагментное, но такие функции нам пригодятся на перспективу.

Листинг 13.92. Функция ru-menu-load-partial

```
(defun ru-menu-load-partial (menu_file_name group_name)
;|-------------------------------------------------------------------------
 Загрузка файла фрагментного меню
 Пример: (ru-menu-load-partial "ruLib" "ruLib")
-------------------------------------------------------------------------|;
 (if (not (menugroup group_name))
 (if (findfile (strcat menu_file_name ".mnu"))
 (progn
  (command "_.MENULOAD" menu_file_name)
  (menucmd (strcat "P" (itoa (1+ (_ru-menu-get-max-pulldown-number)))
 "=+" group_name ".POP1"))
```

```
 (princ (strcat "\nМеню " menu_file_name ".mnu успешно загружено."))
);_ end of progn
(alert (strcat "Не могу найти " menu_file_name ".mnu"))
);_ end of if
 (princ (strcat "\nМеню " menu_file_name ".mnu уже загружено"))
);_ end of if
(princ)
);_ end of defun
```

В отличие от некоторых "бесцеремонных" программ, заталкивающих фрагментные меню в облюбованную позицию, мы предусмотрим вычисление номера последнего падающего меню (листинг 13.93) и будем загружать свои меню в конец линейки. Сгоряча мы можем и стандартное меню загрузить в виде фрагментного:

```
(ru-menu-load-partial "acad" "acad"),
```

но ничего хорошего из этого не получится, т. к. появятся дублированные панели инструментов, второй пункт меню **Файлы** — в конце строки меню.

Листинг 13.93. Функция `_ru-menu-get-max-pulldown-number`

```
(defun _ru-menu-get-max-pulldown-number (/ counter)
;|-------------------------------------------------------------------------
 Определение номера последнего падающего меню
-------------------------------------------------------------------------|;
(setq counter 1)
(while (menucmd (strcat "P" (itoa counter) ".1=?"))
(setq counter (1+ counter))
);_ end of while
(1- counter)
);_ end of defun
```

Листинг 13.94. Функция `ru-menu-unload-partial`

```
(defun ru-menu-unload-partial (group_name)
;|-------------------------------------------------------------------------
Выгрузка фрагментного меню
-------------------------------------------------------------------------|;
 (if (menugroup group_name)(command "_.MENUUNLOAD" group_name))
(princ)
);_ end of defun
```

Функции для управления доступом к меню

В *главе 8* мы писали о необходимости программного управления доступом к пунктам меню. Напоминаем, что мы договорились все свои пункты меню в исходном тексте сделать недоступными. Доступными они могут быть только при загрузке библиотеки ruCAD. В функции инициализации системы и будет устанавливаться доступ ко всем нашим пунктам, теги для которых мы будем начинать с символов RU_. Давайте реализуем эти функции. Сначала напишем простые функции для управления доступом к одиночному пункту меню. Они основаны на использовании функции

menucmd. Недостаток этих функций в том, что надо знать имя группы меню и идентификатор пункта (листинги 13.95—13.99).

Листинг 13.95. Функция ru-menu-tag-disable

```
(defun ru-menu-tag-disable (menu_group tag_menu_item)
;|-------------------------------------------------------------------------
 Запрет доступа к пункту меню
 Срабатывает, если нет DIESEL в меню!!
 Срабатывает только для первого такого ID в файле
 Пример: (ru-menu-id-disable "ruCAD" "RU_MENU_RELOAD")
-------------------------------------------------------------------------|;
(menucmd (strcat "G" menu_group "." tag_menu_item "=~"))
);_ end of defun
```

Листинг 13.96. Функция ru-menu-tag-enable

```
 (defun ru-menu-tag-enable (menu_group tag_menu_item)
;|-------------------------------------------------------------------------
 Разрешение доступа к пункту меню
 Пример: (ru-menu-id-enable "ruCAD"  "ID_HELP")
-------------------------------------------------------------------------|;
(menucmd (strcat "G" menu_group "." tag_menu_item "="))
);_ end of defun
```

Листинг 13.97. Функция ru-menu-tag-check

```
(defun ru-menu-tag-check (menu_group tag_menu_item)
;|-------------------------------------------------------------------------
 Пометка пункта меню галочкой
 Пример: (ru-menu-id-check "ruCAD"  "RU_ID_XML_MENU")
-------------------------------------------------------------------------|;
(menucmd (strcat "G" menu_group "." tag_menu_item "=!"))
);_ end of defun
```

Листинг 13.98. Функция ru-menu-tag-check-rucad-workgroup

```
(defun ru-menu-tag-check-rucad-workgroup ()
;|-------------------------------------------------------------------------
 Пометка галочкой пункта "своей" рабочей группы в сборнике спецменю
-------------------------------------------------------------------------|;
(ru-menu-tag-check "ruCAD" (strcat "RU_ID_" (ru-user-group)))
);_ end of defun
```

Листинг 13.99. Функция ru-menu-tag-uncheck-rucad-workgroup

```
(defun ru-menu-tag-uncheck-rucad-workgroup ()
;|-------------------------------------------------------------------------
 Снятие пометки пункта "своей" рабочей группы в сборнике спецменю
-------------------------------------------------------------------------|;
 (ru-menu-tag-enable "ruCAD" (strcat "RU_ID_" (ru-user-group)))
);_ end of defun
```

А теперь напишем более сложные функции для управления доступом к меню через объектную модель AutoCAD. В этих функциях мы будем использовать параметр tag_wcmatch_pattern. Это строка шаблона, используемого в стандартной функции wcmatch. Функции будут применяться ко всем пунктам падающих и контекстных меню, тег которых соответствует шаблону. Правила формирования шаблона смотрите в справке по функции wcmatch. Чаще всего мы будем применять самые простые шаблоны, но иногда, для того чтобы реже вызывать эти медленные функции, будем использовать составной шаблон.

Начнем с самой главной функции установки доступа (листинг 13.100).

Листинг 13.100. Функция ru-menu-set-enabled

```
(defun ru-menu-set-enabled (reenable / enable_pattern disable_pattern
 comma_enable comma_disable rights_info lst_values lst_names  name i)
;|-------------------------------------------------------------------
Устанавливает режим доступа ко всем пунктам меню.
Аргумент:
reenable - принудительно запускать установку доступа, иначе - в зависимости
от состояния глобальной переменной *ru_menu_menu_loaded* - количества  загрузок меню
-------------------------------------------------------------------|;
;;;-------- Локальная функция -----------------------------
  (defun _ru-menu-set-comma (s)
    (if (= s "") "" ",")
 );_ end of defun
  (if (not reenable)
;;;-------- Главная функция -----------------------------
    (setq reenable (= 0
;|-------------------------------------------------------------------
Проверяем количество загрузок меню, сохраненное во внедокументной переменной
*ru_bb_menu_loaded*
-------------------------------------------------------------------|;
           (vl-bb-ref
             '*ru_bb_menu_loaded*
          );_ end of vl-bb-ref
         )
   );_ end of setq
 );_ end of if
  (if reenable
    (progn
;|-------------------------------------------------------------------
Сначала мы устанавливаем шаблон для разрешения ко всем пунктам меню, у которых тег
начинается с символов RU_, а затем, в зависимости от привилегий пользователя, будем
запрещать доступ к отдельным пунктам
-------------------------------------------------------------------|;
      (setq enable_pattern  "" disable_pattern "" comma_enable ""
            comma_disable "" enable_pattern
          (strcat enable_pattern (_ru-menu-set-comma enable_pattern)
                                   "RU_*");_ end of strcat
     );_ end of setq
      (if (setq rights_info (ru-user-get-rights-info nil))
```

```
        (progn
          (foreach right_lst rights_info
              (setq lst_names (cons (nth 1 right_lst) lst_names)
               lst_values
                     (cons
                       (ru-match-is-bit-in-flag
                         (nth 0 right_lst)
                         (ru-user-right-flag)
                       );_ end of ru-match-is-bit-in-flag
                       lst_values
                     );_ end of cons
              );_ end of setq
          );_ end of foreach
          (setq i 0)
          (foreach value lst_values
            (progn
              (setq name (nth i lst_names))
              (if (not value)
                (setq disable_pattern
                       (strcat disable_pattern
                               (_ru-menu-set-comma
                                 disable_pattern
                               );_ end of _ru-menu-set-comma
                               "RU_"
                               name
                               "*"
                       );_ end of strcat
                );_ end of setq
              );_ end of if
              (setq i (1+ i))
            );_ end of progn
          );_ end of foreach

        );_ end of progn
      );_ end of if
;|-------------------------------------------------------------------------
Доступ к пунктам меню с командами библиотеки Express, включенными в наше меню,
устанавливается, если эта библиотека загружена
-------------------------------------------------------------------------|;
      (if (ru-express-is-loaded)
         (setq enable_pattern
                (strcat enable_pattern
                        (_ru-menu-set-comma enable_pattern)
                        "ID_ACET*"
                );_ end of strcat
         );_ end of setq
         (setq disable_pattern
                (strcat disable_pattern
                        (_ru-menu-set-comma disable_pattern)
                        "ID_ACET*"
                );_ end of strcat
```

```
       );_ end of setq
     );_ end of if
      (setq *ru_menu_loaded* (1+ *ru_menu_loaded*))
;|-------------------------------------------------------------------------
Запоминаем во внедокументной переменной количество загрузок меню. Запоминаем число,
а не простое логическое значение по причинам, о которых вы узнаете в конце раздела
-------------------------------------------------------------------------|;
      (vl-bb-set '*ru_bb_menu_loaded*
                 *ru_menu_loaded*
     );_ end of vl-bb-set
     ;; а теперь разом устанавливаем доступ
      (ru-menu-all-items-action
        (list
          (list enable_pattern '_ru-menu-popupmenuitem-enable nil)
          (list disable_pattern '_ru-menu-popupmenuitem-disable nil)
       );_ end of list
     );_ end of ru-menu-all-items-action
   );_ end of progn
 );_ end of if
 (princ "\nУстановлен доступ к меню\n")
(princ)
);_ end of defun
```

Далее без особых комментариев приведем тексты вспомогательных функций (листинги 13.101—13.113). Надеемся, что читатели сами легко с ними разберутся, обратив внимание на рекурсивные функции.

Листинг 13.101. Функция ru-menu-all-items-action

```
(defun ru-menu-all-items-action (param_list / result)
;| Установка доступа ко всем пунктам меню
Примеры:
Запретить доступ ко всем пунктам "ID_ACET*"
Установить доступ ко всем пунктам "ID_ACET*"
Отметить пункты  "ID_DIM*" "RU_XML*"
(ru-menu-all-items-action
 (list
  (list "RU_*" '_ru-menu-popupmenuitem-disable NIL)
  (list "ID_ACET*" '_ru-menu-popupmenuitem-enable NIL)
  (list "RU_XML*,ID_DIM*" '_ru-menu-popupmenuitem-check NIL)
 )
)
Установить доступ ко всем пунктам  "RU_*"
Снять отметку с пунктов "ID_DIM*" "RU_XML*"
(ru-menu-all-items-action
 (list
  (list "RU_*" '_ru-menu-popupmenuitem-enable nil)
  (list "RU_XML*,ID_DIM*" '_ru-menu-popupmenuitem-uncheck nil)
 )
)
|;
```

```
  (princ "\nЖдите, устанавливаю доступ к меню\n")
  (foreach menu_group (_ru-menu-menugroups-list)
   ;; В каждой MenuGroup
    (foreach popup_menu
                        (_ru-menu-popup-menus-list menu_group)
     ;;В каждом именованном PopupMenu
      (setq result
             (_ru-menu-pop-menu-action
               (_ru-menu-get-popup-menu-by-name menu_group popup_menu)
              param_list result
            );_ end of _ru-menu-pop-menu-action
     );_ end of setq
   );_ end of foreach
 );_ end of foreach
  (reverse result)
);_ end of defun
```

Листинг 13.102. Функция _ru-menu-popupmenuitem-disable

```
(defun _ru-menu-popupmenuitem-disable (popup_menu_item_obj)
;;; Запрещает доступ к объекту пункта меню
  (vlax-put-property popup_menu_item_obj 'Enable :vlax-false)
)
```

Листинг 13.103. Функция _ru-menu-popupmenuitem-enable

```
defun _ru-menu-popupmenuitem-enable (popup_menu_item_obj)
;;; Разрешает доступ к объекту пункта меню
  (vlax-put-property popup_menu_item_obj 'Enable :vlax-true)
)
```

Листинг 13.104. Функция _ru-menu-popupmenuitem-check

```
(defun _ru-menu-popupmenuitem-check (popup_menu_item_obj)
  (vlax-put-property popup_menu_item_obj 'Check :vlax-true)
)
```

Листинг 13.105. Функция _ru-menu-popupmenuitem-uncheck

```
(defun _ru-menu-popupmenuitem-uncheck (popup_menu_item_obj)
  (vlax-put-property popup_menu_item_obj 'Check :vlax-false)
)
```

Листинг 13.106. Функция _ru-menu-menugroups-list

```
(defun _ru-menu-menugroups-list ()
  (ru-obj-list-collection-member-names
    (_ru-menu-get-menugroups-collection)
 )
)
```

Листинг 13.107. Функция _ru-menu-popup-menus-list

```
(defun _ru-menu-popup-menus-list (group_name / popup_menus result)
  (if (setq popup_menus (_ru-menu-get-popup-menus group_name))
    (vlax-for each_popup_menu popup_menus
      (setq result (cons (vla-get-name each_popup_menu) result))
   );_ end of vlax-for
 );_ end of if
  result
)
```

Листинг 13.108. Функция _ru-menu-pop-menu-action

```
(defun _ru-menu-pop-menu-action (popup_menu_obj param_list result_list
                                                          / i)
  (setq i 0)
  (repeat (vla-get-count popup_menu_obj)
    (setq result_list
           (_ru-menu-pop-menu-item-action
             (vla-item popup_menu_obj i)
            param_list
            result_list
          );_ end of _ru-menu-pop-menu-item-action
          i (1+ i)
   );_ end of setq
 );_ end of repeat
  result_list
)
```

Листинг 13.109. Функция _ru-menu-get-popup-menu-by-name

```
(defun _ru-menu-get-popup-menu-by-name (group_name popup_menu_name)
  (vla-item (_ru-menu-get-popup-menus group_name) popup_menu_name)
)
```

Листинг 13.110. Функция _ru-menu-pop-menu-item-action

```
(defun _ru-menu-pop-menu-item-action (popup_menu_item_obj param_list
  result_list / tag_string popup_submenu_obj comparison_function
comparison_function_param label tag_wcmatch_pattern item_result)
;;; вот здесь и обрабатываем каждый пункт PopupMenuItem
  (setq label             (vla-get-label popup_menu_item_obj)
        tag_string        (strcase (vla-get-tagstring
                                       popup_menu_item_obj))
;|-----------------------------------------------------------------
 Проверяем наличие субменю. Если просто применить
(setq popup_submenu_obj (vla-get-submenu popup_menu_item_obj)),
 то на первом же пункте, не имеющем субменю, возникнет ошибка автоматизации,
поэтому подключаем функцию vl-catch-all-apply, которой в качестве параметров
передаем имя метода, который мы пытаемся применить,
и список аргументов, которые надо передать методу, если он сможет выполниться
-----------------------------------------------------------------|;
```

```
        popup_submenu_obj
                          (vl-catch-all-apply
                            'vla-get-submenu
                            (list popup_menu_item_obj)
                          );_ end of vl-catch-all-apply
 );_ end of setq
  (if (not (vl-catch-all-error-p popup_submenu_obj))
    (setq result_list
           (_ru-menu-pop-menu-action
             popup_submenu_obj
             param_list
             result_list
          );_ end of _ru-menu-pop-menu-action
   );_ end of setq
 );_ end of if
  (foreach param param_list
    (progn
      (setq tag_wcmatch_pattern (strcase (nth 0 param))
            comparison_function (nth 1 param)
            comparison_function_param (nth 2 param)
     );_ end of setq
      (if (wcmatch tag_string tag_wcmatch_pattern)
        (progn
          (if comparison_function_param
            ((eval comparison_function)
              popup_menu_item_obj
              comparison_function_param
            )
            ((eval comparison_function) popup_menu_item_obj)
         );_ end of if
          (setq result_list
                 (cons (reverse
                         (cons comparison_function_param
                               (cons comparison_function
                                     (cons tag_string item_result)
                               );_ end of cons
                         );_ end of cons
                       );_ end of reverse
                       result_list
                 );_ end of cons
         );_ end of setq
       );_ end of progn
     );_ end of if
   );_ end of progn
 );_ end of foreach
  result_list
);_ end of defun
```

Листинг 13.111. Функция _ru-menu-get-menugroups-collection

```
(defun _ru-menu-get-menugroups-collection ()
    (ru-obj-acad-collection "MenuGroups")
);_ end of defun
```

Листинг 13.112. Функция _ru-menu-get-popup-menus

```
(defun _ru-menu-get-popup-menus (group_name / menu_group)
  (if (setq menu_group (_ru-menu-get-menugroup group_name))
    (vla-get-menus menu_group)
    (princ (strcat "\nНе найдена группа меню: " group_name))
 );_ end of if
)
```

Листинг 13.113. Функция _ru-menu-get-menugroup

```
(defun _ru-menu-get-menugroup (group_name)
  (if (menugroup group_name)
    (vla-item (_ru-menu-get-menugroups-collection) group_name)
 );_ end of if
)
```

Кошмарная пятница

Вся эта хитрая система работала просто замечательно. Пользователь был огражден от ненужных ему переживаний и доступа к опасным программам, меню само знало, загружена ли библиотека Express, при загрузке нашего меню в систему AutoCAD были доступны только стандартные команды. Но вдруг (разумеется, в пятницу, и, разумеется, 13-го числа) случилось страшное. Обнаружилось, что *после перегрузки файла меню* стала "рушиться" система AutoCAD. При первоначальной загрузке, даже неоткомпилированного файла меню, все было нормально, а вот при *повторной загрузке файла меню* в одном сеансе система AutoCAD выдавала сообщение *"cannot allocate memory - 0xC0000005: Access Violation"*, запрашивала отладчик и через некоторое время прекращала свою жизнедеятельность. Многократное (сотни раз) выполнение выражения `(ru-menu-set-enabled T)` ошибок не вызывало, т. е. сами функции вроде бы написаны корректно.

Разумеется, мы стали искать собственные ошибки — у кого же их нет?! Описывать, где и как мы их искали, не будем — на это ушло два человеко-месяца. Мы перепробовали все, но на всех доступных нам компьютерах, на всех версиях системы AutoCAD, на разных меню — везде повторялась одна и та же ошибка, впрочем иногда коварно исчезая на несколько запусков. Для того чтобы случайно не повторить возможные ошибки, был написан совсем другой код, выполняющий аналогичные действия, но ошибка осталась.

Управление меню мы осуществляли через объекты. Везде, где только можно, были расставлены ловушки ошибок. Если бы ошибка происходила на уровне COM-интерфейсов, она была бы изловлена. Так как постоянно появлялась типично связанная с языком C ошибка, да еще такая, которая может возникнуть по множеству причин, чаще всего из-за несбалансированности создания и уничтожения переменных, мы в конце концов решили, что ошибка находится где-то внутри самой системы AutoCAD.

Как придется выкручиваться

Дальнейшее ковыряние в этой проблеме не имело смысла. Мы ведь делаем реальную систему в реальные сроки (и только попутно пишем книгу), которые нельзя

затягивать из-за одного сомнительного места. Пришлось вернуться к отработанной схеме управления доступом с использованием языка DIESEL. Пункты меню были вновь защищены "старинным" способом:

```
[$(if,$(eq,$(substr,$(getvar,SECRET),$(-, $(strlen, $(getvar, SECRET)),
$(strlen, _RUCAD))),_RUCAD),~)Пункт меню]
```

В данном варианте проверяется загрузка функций нашей системы по наличию в конце некой строковой системной переменной SECRET сигнатуры _RUCAD.

Проверка загрузки библиотеки Express Tools осуществляется выражением

```
[$(if,$(eq,$(substr,$(getvar, SECRET),21,1),0),~)Express]
```

которое делает недоступным пункт меню, если в 21-й позиции переменной SECRET находится 0.

Доступ к опасным пунктам меню блокируется выражением

```
[$(if,$(eq,$(substr,$(getvar, SECRET),X,1),0),~)Для профи]
```

которое делает недоступным пункт меню, если в X-й позиции переменной SECRET находится 0.

Управление системной переменной SECRET осуществляет функция `ru-menu-diesel` (листинг 13.114). Она формирует строку примерно такого вида: 1111110110111011110001001_RUCAD. Символы 1 или 0 выставляются в зависимости от результатов проверки прав пользователя функцией `ru-user-test-permission`. Данная функция на этапе разработки много раз изменялась, условно-окончательный вариант для книжной версии определится только в *главе 36*. Проверяется также состояние некоторых глобальных переменных (для установки "галочек" в меню) и загрузка библиотеки Express Tools.

Листинг 13.114. Функция `ru-menu-diesel`

```
(defun ru-menu-diesel ()
;;; установка строки доступа к меню
  (defun _set_var (value)
    (setvar "SECRET" (strcat (getvar " SECRET ") (if value "1" "0")))
 );_ end of defun

  (setvar "SECRET" "")
  (mapcar '_set_var
    (mapcar 'ru-user-test-permission
     (ru-user-get-permissions-names-list))
 );_ end of mapcar
  (repeat (- 20 (strlen (getvar "SECRET"))) (_set_var nil))
 ;; добавляем другие признаки
 ;;Проверка наличия Express
 ;; позиция 21
  (_set_var (ru-express-is-loaded))
 ;; позиция 22
  (_set_var *ru_msg_debug*)
 ;; позиция 23
  (_set_var *ru_previewhelp*)
```

```
 ;; позиция 24
  (_set_var *ru_xml_edit*)
 ;; позиция 25
  (_set_var *ru_developer*)
  (setvar "SECRET" (strcat (getvar "SECRET") "_RUCAD"))
);_ end of defun
```

А что это за переменная SECRET? Нет такой системной переменной в системе AutoCAD!

Таким псевдонимом мы обозначили некую переменную, которая должна удовлетворять следующим условиям:

- быть системной переменной AutoCAD;
- иметь строковый тип;
- никогда не использоваться или очень редко использоваться иными приложениями;
- не относиться к USERS1—USERS5.

Ради любопытства попробуйте высчитать подлинное имя этой переменной.

Замечание

Очень неудобно, что в языке DIESEL можно использовать только системные переменные. Разработчики фирмы Autodesk могли бы предусмотреть возможность использования в выражениях DIESEL глобальных переменных или какой-то иной способ чтения личных переменных пользователя. Но не хотят.

Функцию `ru-menu-diesel` мы должны выполнять каждый раз при изменении контролируемых ей условий.

Ввод данных

Организация ввода данных очень важна для любой программы. В *главе 10* мы писали, что большинство ошибок может возникнуть именно на этапе ввода данных пользователем. Ошибки вычислительного этапа программист может и обязан предотвратить, но если он не локализует ошибки ввода, все остальные усилия будут напрасными. К сожалению, очень часто программы пишут так, что их нормальная работа просто невозможна при любом отклонении от задуманного программистом сценария. Об этом мы достаточно подробно писали в *главе 10*. Теперь нам предстоит реализовать собственные функции, максимально предотвращающие возможность ошибок на этапе ввода данных. AutoLISP имеет встроенные функции для ввода всех типов данных, но возможностей их явно недостаточно. Функции ввода данных должны:

- выводить как можно более информативное приглашение, максимально приближенное к принятому в AutoCAD стандарту;
- позволять ввести значение по умолчанию, если это допускается логикой приложения;
- блокировать попытки ввода данных недопустимых типов или вне допустимого диапазона значений;

- блокировать ошибочные действия пользователя;
- позволять пользователю воспользоваться дополнительными удобствами в виде контекстных и иных меню для максимального сокращения клавиатурного ввода.

Информативность сообщения предполагает, что пользователь должен видеть в командной строке четкое и недвусмысленное приглашение, *все* возможные опции, значение по умолчанию, записанное в угловых скобках, и *обязательно* — штатную возможность выхода (если допустимо на данном этапе, то и досрочного выхода). Пользователь не должен раздумывать, как ему выйти из команды — то ли нажать <Enter>, то ли <Esc>, то ли <Ctrl>+<Alt>+<Del>.

Примечание

Особенно ужасно, когда пользователя заставляют последовательно выполнять обязательные (иногда десятки) действия для доведения программы до конца. Такие программы до сих пор встречаются, хотя событийно-ориентированный интерфейс Win-приложений все-таки дает пример самым "упертым" программистам.

Стандартные команды системы AutoCAD также не всегда являются образцом. Например, почти у всех команд не отображается действие по умолчанию (при нажатии клавиши <Enter>).

Простая команда LINE (ОТРЕЗОК) выводит неполные сообщения. При запросе первой точки имеется возможность, нажав клавишу <Enter>, продолжить рисование от предыдущей точки, но пользователь об этом не извещается:

LINE Specify first point:

А должно бы быть наподобие:

LINE Specify first point ***<Предыдущая точка>:***

При вводе последующих точек:

Specify next point or [Undo]:
Specify next point or [Close/Undo]:

А надо бы:

Specify next point or [Undo] ***<Выход>:***
Specify next point or [Close/Undo] ***<Выход>:***

Если для стандартной команды лаконичное сообщение *"Specify next point"* является правильным, то в прикладной программе этого явно недостаточно. Представьте, что в программе рисования прямоугольника пользователь получит запросы (в скобках мысли и действия "тетки"):

Укажите точку: (Какую точку? Угол? Какой из четырех? Или центр? Дай-ка ткну наугад!)
Укажите точку: (А теперь какую точку? Ничего не понятно! Опять ткну наугад!)
Укажите точку: (Достали!!!)
Укажите угол: (Какой еще угол?)

Конечно, после ряда "дрессировок" даже "обезьяна с гранатой" сможет нарисовать прямоугольник, но можно было бы сделать сразу понятно:

РИСОВАНИЕ ПРЯМОУГОЛЬНИКА ПО ЦЕНТРУ И РАЗМЕРАМ СТОРОН:
Длина=100, Ширина=80. Укажите центр прямоугольника [Длина/Ширина] <Выход>:(Вводит Д или выбирает из контекстного меню Длина)

Длина прямоугольника [Справочник] <100>:(Выбирает из контекстного меню Справочник и в диалоговом окне выбирает 200)
Длина=400, Ширина=80. Укажите центр прямоугольника [Длина/Ширина] <Выход>:(Вводит Ш или выбирает из контекстного меню Ширина)

Ширина прямоугольника [Справочник] <80>: (Вводит 300)
Длина=400, Ширина=300. Укажите центр прямоугольника [Длина/Ширина] <Выход>: (Указывает 0,0)
Угол поворота прямоугольника<0>: (Вводит 45 или поворачивает мышью)
Длина=400, Ширина=300. Укажите центр прямоугольника [Длина/Ширина] <Выход>: (Жмет <Enter> для завершения программы)

При такой организации ввода пользователь знает, что он рисует, имеет размеры прямоугольника по умолчанию (велика вероятность, что ему надо рисовать много одинаковых прямоугольников), может (но не обязан) изменить любой размер, в любой допустимый момент штатно выйти из программы, при вводе размеров выбрать их из справочника (или ввести с клавиатуры). После завершения рисования программа предложит повторить с такими же размерами. Прекратить цикл "рисований" нажатием клавиши <Enter> проще, чем вновь выбирать программу в меню.

При небольшом усовершенствовании — выносе текста приглашения и файла справочника в аргументы функции — мы можем получить сразу множество специализированных программ. Вентиляционник будет рисовать "Разрез воздуховода", электрик — "Щит", а строитель — "Колонну" или "Стол". Практика показывает, что пользователям больше нравятся специализированные программы, чем абстрактные прямоугольники, хотя и такой пункт в меню должен присутствовать.

Реализацию всех принципов надежного ввода данных мы рассмотрим в исходных текстах, не задерживаясь более на теоретические дискуссии.

Традиционные функции ввода данных

Многие программисты применяют улучшенные функции ввода данных, разработанные по схеме, приведенной в листинге 13.115.

Листинг 13.115. Функция `MyGetDist`

```
(defun MyGetDist (msg default base_point / result message)
;| Примеры:
   (MyGetDist "Длина прямоугольника" 400 (getvar "LASTPOINT"))
   (MyGetDist "Длина прямоугольника" 400 nil)
|;
  (setq message (strcat "\n" msg " <" (rtos default) ">: "))
  (initget 4)
  (if base_point
```

```
    (setq result (getdist base_point message))
    (setq result (getdist message))
 );_ end of if
  (if result result default)
);_ end of defun
```

Подобные функции применяются и для перекрытия остальных базовых функций — getint, getreal, getpoint, getangle. При использовании приведенной функции пользователь может ввести значение указанием (от заданной базовой точки или двумя точками) с клавиатуры или принять предлагаемое значение по умолчанию. Это хорошо, но мало. При этом нельзя передать опции ввода (например, дать возможность выбора из справочника), и не предотвращается прерывание при нажатии клавиши <Esc>, приводящее к прерыванию всей программы, а не только выполняющейся функции.

Попробуем все усовершенствовать.

Усовершенствованные функции ввода данных

Начнем с универсальной низкоуровневой функции, на которой будет основан весь ввод данных (листинг 13.116).

Листинг 13.116. Функция _ru-get-with-default

```
(defun _ru-get-with-default (message default_str quoted_get_func
 initget_param keywords base_point / result question lst_params
 key_str)
;|-------------------------------------------------------------------------
Действие:
Выполняет функцию quoted_get_func с возможностью получения значения
по умолчанию
Аргументы:
message - строка запроса
default_str - строка, выводящаяся в <default_str>
quoted_get_func - имя getxxx-функции с апострофом
initget_param - биты для initget или nil
keywords - ключевые слова для initget или nil
base_point - базовая точка или nil
Возвращает значение, ключевое слово или nil
Не позволяет прервать ввод нажатием клавиши Esc
-------------------------------------------------------------------------|;
 (setq key_str "")
 (if keywords
;; Формирование строки ключевых слов
  (progn
      (setq lst (ru-string-to-list keywords " "))
      (foreach word lst
        (setq key_str (strcat key_str
                              word
                              (if (/= word (last lst))
```

```
                                    "/"
                                    ""
                                );_ end of if
                            );_ end of strcat
        );_ end of setq
      );_ end of foreach
       (setq key_str (ru-string-format "[%1]" key_str))
    );_ end of progn
  );_ end of if
;; Формирование строки запроса
   (if default_str
     (setq question (ru-string-format
                      "\n%1 или %2<%3>:"
                      (list message key_str default_str)
                    );_ end of ru-string-format
    );_ end of setq
     (setq question
            (ru-string-format "\n%1 или %2:" (list message key_str))
    );_ end of setq
  );_ end of if
   (if base_point
     (setq lst_params (list base_point question))
     (setq lst_params (list question))
  );_ end of if
;; Выполнение заданной функции с блокировкой возможных ошибок
   (while
     (vl-catch-all-error-p
       (progn
         (cond ((and initget_param keywords)
                (initget initget_param keywords)
               )
               ((not (null initget_param))
                (initget initget_param)
               )
               ((not (null keywords))
                (initget keywords)
               )
         );_ end of cond
         (setq result (vl-catch-all-apply quoted_get_func lst_params))
      );_ end of progn
    );_ end of vl-Catch-All-Error-P
  );_ end of while
   result
);_ end of defun
```

Разобраться, как работает эта функция, можно только путем тщательного анализа кода и примеров использования. Функция достаточно сложна в применении, поэтому мы в дальнейшем "обернем" ее высокоуровневыми функциями для использования в прикладных программах. Начнем с ввода дистанции (листинг 13.117).

Листинг 13.117. Функция `ru-get-dist`

```
(defun ru-get-dist  (message default base_point)
;|-------------------------------------------------------------------------
Примеры:
(ru-get-dist "Длина трассы" 100 nil)
(ru-get-dist "Длина трассы" 100 (list 0.0 0.0))
-------------------------------------------------------------------------|;
 (cond
 ((_ru-get-with-default  message (rtos default) 'getdist 4 nil
   base_point))
   (T default)
)
)
```

Визуально работа функции `ru-get-dist` не отличается от работы `MyGetDist`, за исключением маленькой, но чрезвычайно важной детали — пользователь не может прервать ввод нажатием клавиши <Esc>.

А если по логике программы нужно иметь возможность выхода? Для такой распространенной ситуации мы разработаем еще одну функцию (листинг 13.118).

Листинг 13.118. Функция `ru-get-dist-or-exit`

```
(defun ru-get-dist-or-exit (message default base_point / dist)
;|-------------------------------------------------------------------------
Примеры:
(ru-get-dist-or-exit "Длина трассы" 100 nil)
(ru-get-dist-or-exit "Длина трассы" 100 (list 0.0 0.0))
-------------------------------------------------------------------------|;
(setq dist (_ru-get-with-default message (rtos default) 'getdist
 4 "Выход" base_point))
 (cond ((= dist "Выход") nil)
       ((null dist) default)
       (T dist)
 );_ end of cond
);_ end of defun
```

Теперь во время ввода пользователь имеет опцию "Выход" и при ее выборе, в том числе из невесть откуда взявшегося контекстного меню, функция вернет `nil`, во всех остальных случаях будет возвращено значение дистанции.

```
Command: (ru-get-dist-or-exit "Длина трассы" 100 nil)
Длина трассы или [Выход]<100>:
```

Теперь разработаем функцию для ввода целого числа (листинг 13.119).

Листинг 13.119. Функция ru-get-int

```
(defun ru-get-int (message default)
;|-------------------------------------------------------------------------
Примеры:
(ru-get-int "Количество колонн" 3)
(ru-get-int "Количество колонн" nil)
-------------------------------------------------------------------------|;
(cond
  ((_ru-get-with-default message
    (if default (rtos default) nil) 'getint (if default 4 5) nil nil)
 )
  (T default)
);_ end of cond
)
```

Для ввода положительных вещественных чисел (необходимость в отрицательных числах у нас просто не возникала) разработаем следующую функцию (листинг 13.120).

Листинг 13.120. Функция ru-get-real-positive

```
(defun ru-get-real-positive (message default)
;|-------------------------------------------------------------------------
Примеры:
(ru-get-real-positive "Ширина трассы" 100)
(ru-get-real-positive "Ширина трассы" nil)
-------------------------------------------------------------------------|;
  (cond
   ((_ru-get-with-default message (if default (rtos default) nil)'getreal
       (if default 4  5) nil nil)
  )
   (T default)
 );_ end of cond
);_ end of defun
```

Ввод точек у нас будет осуществляться очень часто и в различных вариантах. Придется разработать еще несколько функций (листинги 13.121—13.127).

Листинг 13.121. Функция ru-get-point-required

```
(defun ru-get-point-required  (message  base_point)
;|-------------------------------------------------------------------------
 Запрос ввода точки с обязательным указанием
(ru-get-point-required "Вторая точка" (list 10.0 10.0))
(ru-get-point-required "Вторая точка"  nil)
-------------------------------------------------------------------------|;
 (_ru-get-with-default  message nil  'getpoint 1 nil base_point)
)
```

Листинг 13.122. Функция `ru-get-point-with-default`

```
(defun ru-get-point-with-default  (message default base_point)
;|-------------------------------------------------------------------------
Запрос ввода точки с не обязательным указанием и значением по умолчанию
(ru-get-point-with-default "Вторая точка" (list 10.0 10.0) (list 0.0 0.0))
(ru-get-point-with-default "Вторая точка" (list 10.0 10.0) nil)
-------------------------------------------------------------------------|;
(cond
  ((_ru-get-with-default message
     (ru-list-to-string (mapcar 'rtos default) ",") 'getpoint nil nil
      base_point)
  )
   (T default)
 )
)
```

Часто потребуется при вводе точки иметь возможность выбора опций (листинг 13.123).

Листинг 13.123. Функция `ru-get-point-or-exit`

```
(defun ru-get-point-or-exit (message keywords / result key_str)
;|-------------------------------------------------------------------------
Примеры:
(ru-get-point-or-exit "Первая точка" "Начало Образец")
(ru-get-point-or-exit "Первая точка" nil)
-------------------------------------------------------------------------|;
  (_ru-get-with-default   message  "Выход"  'getpoint nil keywords nil)
)
```

Во многих программах потребуется ввод точки с опциями изменения веса линии, продолжения и выхода (листинг 13.124).

Листинг 13.124. Функция `ru-get-point-or-lw-or-continue-or-exit`

```
(defun ru-get-point-or-lw-or-continue-or-exit (msg / point)
;|-------------------------------------------------------------------------
Пример:
 (ru-get-point-or-lw-or-continue-or-exit "Начало патрубка")
-------------------------------------------------------------------------|;
  (setq point (ru-get-point-or-exit msg "ВЕСлинии Далее"))
  (cond
    ((= point "ВЕСлинии")
     (ru-dlg-dcl-select-lw);; Диалог выбора веса линии
     (setvar "CELWEIGHT" (ru-lw-current))
   )
    ((= point "Далее");; Продолжение от точки, запомненной в глобальной
     (if *ru_last_end_point*
```

```
      (setq point *ru_last_end_point*)
      (princ "\nПредыдущей точки нет!")
    );_ end of if
   )
 );_ end of cond
  point
);_ end of defun
```

Не лишней будет и функция запроса точки с построением резинового прямоугольника от базовой точки (листинг 13.125).

Листинг 13.125. Функция ru-get-corner-required

```
(defun ru-get-corner-required (message base_point)
;|-------------------------------------------------------------------------
Пример:
(ru-get-corner-required "Вторая точка" (list 10.0 10.0))
-------------------------------------------------------------------------|;
  (_ru-get-with-default message nil 'getcorner 1 nil base_point)
);_ end of defun
```

Не избежать нам и ввода ключевых слов (листинг 13.126).

Листинг 13.126. Функция ru-get-kword

```
(defun ru-get-kword (message keywords default)
;|-------------------------------------------------------------------------
Пример:
(ru-get-kword "Подумай и ответь" "Да Нет" "Да")
-------------------------------------------------------------------------|;
(cond
  ((_ru-get-with-default  message default 'getkword nil keywords nil))
  (T default)
 )
);_ end of defun
```

Рано или поздно потребуется ввод углов (листинг 13.127).

Листинг 13.127. Функция ru-get-angle

```
(defun ru-get-angle (message default base_point / result question)
;|-------------------------------------------------------------------------
Пример:
(ru-get-angle "Угол поворота формата" 0 (list 0.0 0.0))
-------------------------------------------------------------------------|;
(cond
  ((_ru-get-with-default message (angtos default) 'getangle nil nil
     base_point)
 )
```

```
  (T default)
)
);_ end of defun
```

Как видите, имея одну базовую низкоуровневую функцию, мы можем очень легко разработать множество конечных функций любого назначения. Различные более частные варианты мы еще будем разбирать.

Наша функция `_ru-get-with-default` далеко не идеальна. Возможны и более изощренные варианты, но для наших потребностей этого пока достаточно.

Замечание

В приведенных функциях ввода данных таится незаметная, но грубая ошибка — ввод производится без учета активной системы координат. Пока активной является мировая система координат, все будет нормально. Функции ввода основаны на стандартных get-функциях, которые принимают опорные точки в активной системе координат и возвращают введенные точки в этой же системе, а рисование производится с использованием разных систем. Функция `command` использует активную систему координат, а вся работа объектными методами производится только с мировыми координатами. Подробно эта проблема будет расмотрена в *главе 22*.

Функции выбора примитивов

Выбор объектов производится очень часто. О возможных ошибках при операциях выбора мы также писали в *главе 10*. Пора реализовать намеченные решения. Мы уже знакомы с функциями выбора примитива с анализом промаха при указании `ru-ss-entsel` (см. листинг 10.16) и выбора примитива с учетом допустимых типов `ru-ss-entsel-by-type` (см. листинг 10.15). Эти функции являлись усовершенствованными оболочками стандартной функции `entsel`, предназначенной для выбора одного примитива.

Организовать аналогичный механизм можно и с использованием функции `ssget` (листинг 13.128).

Листинг 13.128. Функция `ru-ss-ones-alt`

```
(defun ru-ss-ones-alt (message list_types true_lock /
                                                    sel cmd_lst)
;|-------------------------------------------------------------------
 Аргументы:
 message - краткое приглашение для выбора, допускается nil
 list_types - список имен допустимых типов примитивов, допускается nil
 true_lock - разрешение выбора на заблокированном слое - (T - разрешает, nil -
запрещает)
 Пример вызова:
 (ru-select-ones "Выбери отрезок или полилинию" '("LINE" "LWPOLYLINE")
 nil)
 Возвращает имя первого примитива в наборе при удачном выборе или nil при отказе
с помощью Enter или прерывании по Esc, в последнем случае одновременно выводит
сообщение о прерывании в командную строку.
-------------------------------------------------------------------|;
```

```
  (setq message
        (strcat "\n" (if message (strcat message " ") "") "<Выход>")
         cmd_lst (list (if true_lock "_:S:E" "_:S:E:L"))
         list_types
           (mapcar (function (lambda (x) (cons 0 x))) list_types)
 );_ end of setq
  (if (and list_types (> (length list_types) 1))
    (setq list_types
          (append (cons '(-4 . "<OR") list_types) '((-4 . "OR>")))
   );_ end of setq
 );_ end of if
  (if list_types  (setq cmd_lst (append cmd_lst (list list_types))))
  (setvar "ERRNO" 0)
  (while (and (/= (getvar "ERRNO") 52) (not sel))
    (princ message)
    (vl-catch-all-error-p (setq sel (vl-catch-all-apply 'ssget cmd_lst)))
 );_ end of while
  (cond
    ((not sel) nil)
    ((= (type sel) 'pickset) (ssname sel 0))
    ((= (type sel) 'vl-catch-all-apply-error)
      (princ (vl-catch-all-error-message sel))
       nil
   )
    (t nil)
 )
);_ end of defun
```

Пример:

```
Выбери отрезок или полилинию <Выход>
Select objects:
1 was filtered out.
<Entity name: 40074428>
```

В чем преимущества такого подхода? В первую очередь, мы возлагаем обязанности по фильтрации примитивов на функцию `ssget`, для которой это штатный режим ее работы. Второй плюс в том, что нам нет необходимости обрабатывать причину неудачного выбора, поскольку данная функция сама умеет это делать и оповещает пользователя о причинах неудачи *стандартным* для данной версии и локализации AutoCAD образом.

Однако эта функция имеет существенный недостаток — постороннее сообщение *Select objects:*, выдаваемое стандартной функцией `ssget`, запрятанной глубоко в тело нашей оболочки. Мы предлагаем пользователю выбрать *один объект*, а стандартная функция (о которой "тетке" знать не положено) как бы поправляет нас и говорит, что надо выбрать *объекты*, да еще на отличном от основной программы языке. Оповещение стандартным образом *1 was filtered out* в прикладной программе также может дезориентировать неискушенного пользователя.

Именно подобные мелочи раздражают при работе многих программ. Избавиться от ненужных сообщений можно установкой системной переменной NOMUTT=1.

Системная переменная NOMUTT управляет полным подавлением запросов и сообщений в командной строке. Допустимые значения NOMUTT:

- 0 — нормальный режим подсказок;
- 1 — полное подавление подсказок.

Необходимо только изменить участок кода:

```
  (while (and (/= (getvar "ERRNO") 52) (not sel))
    (princ message)
    (setvar "NOMUTT" 1)
    (vl-catch-all-error-p (setq sel (vl-catch-all-apply 'ssget cmd_lst)))
    (setvar "NOMUTT" 0)
 );_ end of while
```

Сообщения функции `ssget` будут подавляться, а наши — будут выводиться, т. к. их вывод производится раньше отключения сообщений.

Предупреждение

Пользоваться системной переменной NOMUTT нужно очень осторожно, т. к. она подавляет весь вывод. Система AutoCAD нечаянно может оказаться "глухонемой". Именно поэтому мы устанавливаем NOMUTT=0, а не запоминаем и не восстанавливаем предыдущее состояние. Однако взять на заметку эту переменную стоит, т. к. иногда действительно нужно подавить вывод стандартных сообщений, от которых иными способами нельзя избавиться.

В дальнейшем мы все-таки остановимся на вариантах выбора, базирующихся на функции `entsel`. Попробуем сделать аналог универсальной функции ввода данных.

Листинг 13.129. Функция _ru-get-ent-default

```
(defun _ru-get-ent-default (message default_str quoted_get_func
                       / result question lst_params key_str)
(if default_str
  (setq question (strcat "\n" message " <" default_str ">: "))
  (setq question (strcat "\n" message ": "))
);_ end of if
(setq lst_params (list question))
(while
  (vl-catch-all-error-p
    (progn
        (setq result (vl-catch-all-apply quoted_get_func lst_params))
   );_ end of progn
 );_ end of vl-Catch-All-Error-P
 );_ end of while
  result
);_ end of defun
```

Теперь начнем отрабатывать применение этой функции. Сначала предусмотрим блокировку промаха при указании примитива. Функция `_ru-get-entsel-no-error` не позволяет промахнуться, но и не блокирует нажатие клавиши <Esc> (листинг 13.130).

Листинг 13.130. Функция _ru-get-entsel-no-error

```
(defun _ru-get-entsel-no-error (message / ent)
;|-------------------------------------------------------------------------
Как и стандартная функция, возвращает примитив и точку указания.
Не дает промахнуться, но не блокирует ESC
Пример: (_ru-get-entsel-no-error "Выбери объект, но не промахнись!")
-------------------------------------------------------------------------|;
 (setvar "ERRNO" 0)
 (while
    (and
     (not (setq ent (entsel  (strcat "\n" message))))
     (equal 7 (getvar "ERRNO"));;Ошибка указания при выборе
   );_ end of and
    (setvar "ERRNO" 0)
);_ end of while
 (cond
  ((equal (getvar "ERRNO") 52) nil)
  (T ent)
 );_ end of cond
);_ end of defun
```

Эта функция нам нужна для вызова в высокоуровневой функции `ru-get-entsel` (листинг 13.131).

Листинг 13.131. Функция ru-get-entsel

```
(defun ru-get-entsel (message)
;|-------------------------------------------------------------------------
Применяется при отсутствии требований к блокировке и типам примитивов
Возвращает примитив и точку указания
Пример: (ru-get-entsel "Выбери объект, но не промахнись!")
-------------------------------------------------------------------------|;
  (_ru-get-ent-default message "Выход"  '_ru-get-entsel-no-error)
)
```

Аналогичным образом разработаем функции для указания вложенных примитивов (листинги 13.132—13.134).

Листинг 13.132. Функция _ru-get-nentsel-no-error

```
(defun _ru-get-nentsel-no-error (message / ent)
;|-------------------------------------------------------------------------
Пример: (_ru-get-nentsel-no-error "Выбери объект, но не промахнись!")
-------------------------------------------------------------------------|;
  (setvar "ERRNO" 0)
  (while
    (and
      (not (setq ent (nentsel (strcat "\n" message))))
```

```
      (equal 7 (getvar "ERRNO"))
   );_ end of and
    (setvar "ERRNO" 0)
 );_ end of while
  (cond
    ((equal (getvar "ERRNO") 52) nil)
    (T ent)
 );_ end of cond
);_ end of defun
```

Листинг 13.133. Функция `ru-get-nentsel`

```
(defun ru-get-nentsel (message)
;|-------------------------------------------------------------------------
Пример: (ru-get-nentsel "Выбери объект, но не промахнись!")
Применяется при отсутствии требований к блокировке и типам примитивов
Возвращает примитив и точку указания
-------------------------------------------------------------------------|;
  (_ru-get-ent-default message "Выход" '_ru-get-nentsel-no-error)
);_ end of defun
```

Листинг 13.134. Функция `ru-get-entsel-by-type`

```
 (defun ru-get-entsel-by-type (message msg_err_types list_types no_locked
  / ent ent_type bad_type locked do)
;|-------------------------------------------------------------------------
Выбор примитива с воможностью задать допустимые типы и выбор на неблокированном
слое с возможностью выхода и с блокировкой ESC
Аргументы:
message - сообщение
msg_err_types - сообщение о неверном типе, если задан список типов, иначе ""
list_types - список допустимых типов или nil
no_locked  - выбор на неблокированном слое - T, на любом - nil
Примеры:
 (ru-get-entsel-by-type "Выбери отрезок или полилинию" "Это не ОТРЕЗОК и не
ПОЛИЛИНИЯ" (list "LINE" "LWPOLYLINE") T)
 (ru-get-entsel-by-type "Выбери объект на неблокированном слое" "" nil  T)
 (ru-get-entsel-by-type "Выбери отрезок или полилинию - можно на блокированном"
"Это не ОТРЕЗОК и не ПОЛИЛИНИЯ" (list "LINE" "LWPOLYLINE") nil)
Возвращает имя примитива и точку указания или nil при отказе
-------------------------------------------------------------------------|;
  (setq do T)
  (while do
  (setq bad_type T locked T)
  (if (setq ent (ru-get-entsel message))
   (progn
    (setq ent_type (cdr (assoc 0 (entget (car ent)))))
    (if (and list_types (not (member ent_type list_types)))
     (princ (strcat "\nОШИБКА: Указан объект типа '"
        ent_type "'. " msg_err_types))
     (setq bad_type nil)
   );_ end of if
```

```
    (if (and no_locked
          (ru-layer-is-lock (cdr (assoc 8 (entget (car ent)))))
     );_ end of and
     (princ "\nОШИБКА: Объект на заблокированном слое!")
     (setq locked nil)
   );_ end of if
    (setq do (or bad_type locked))
  );_ end of progn
   (setq do nil)
 );_ end of if
);_ end of while
  ent
);_ end of defun
```

Сделаем, без дополнительных пояснений, еще несколько функций для работы с наборами и отдельными примитивами (листинги 13.135—13.149). Примеры их использования встретятся нам в программах.

Листинг 13.135. Функция `ru-ent-dxf-code-data`

```
(defun ru-ent-dxf-code-data (dxf_code lst)
    (cdr (assoc dxf_code lst))
 );_ end of defun
```

Листинг 13.136. Функция `ru-ent-multi-dxf-code-data`

```
(defun ru-ent-multi-dxf-code-data (dxf_code lst / result item)
;|-------------------------------------------------------------------------
Возвращает список значений ВСЕХ точечных пар с заданным кодом
Пример:
Пусть (entget (entlast)) вернет
((-1 . <Entity name: 4009e6c8>) (0 . "LWPOLYLINE") (330 . <Entity name: 4009ccf8>)
(5 . "1A9") (100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "0")
(100 . "AcDbPolyline") (90 . 6) (70 . 128) (43 . 0.0) (38 . 0.0) (39 . 0.0)
(10 -2.79849 -4.63591) (40 . 0.0) (41 . 0.0) (42 . 0.0)
(10 -2.79849 -1.04173) (40 . 0.0) (41 . 0.0) (42 . 0.0)
(10 9.25113 -1.04173) (40 . 0.0) (41 . 0.0) (42 . 0.0)
 (10 9.25113 6.76632) (40 . 0.0) (41 . 0.0) (42 . 0.0) (10 14.9654 6.76632)
(40 . 0.0) (41 . 0.0) (42 . 0.0) (10 14.9654 8.87326) (40 . 0.0)
  (41 . 0.0) (42 . 0.0) (210 0.0 0.0 1.0))
Тогда
 (ru-ent-multi-dxf-code-data 42 (entget (entlast))) вернет
 (0.0 0.0 0.0 0.0 0.0 0.0)
 (ru-ent-multi-dxf-code-data 10 (entget (entlast))) вернет
((-2.79849 -4.63591) (-2.79849 -1.04173) (9.25113 -1.04173) (9.25113 6.76632)
(14.9654 6.76632) (14.9654 8.87326))
Для сравнения
(ru-ent-dxf-code-data 10 (entget (entlast)))
(-2.79849 -4.63591)
вернет только первый элемент
-------------------------------------------------------------------------|;
```

```
  (foreach item lst
    (if (= (car item) dxf_code)
      (setq result (cons (cdr item) result))
   );_ end of if
 );_ end of foreach
  (if result
    (reverse result)
    nil
 );_ end of if
);_ end of defun
```

Листинг 13.137. Функция ru-ent-dxf-code-clear-list

```
(defun ru-ent-dxf-code-clear-list (lst list_dxf_codes is_stay_value)
    (cond
        ((null lst) nil)
        ((/= is_stay_value (= (type (member (caar lst) list_dxf_codes)) 'list))
         (ru-ent-dxf-code-clear-list (cdr lst) list_dxf_codes is_stay_value)
       )
        (T (cons (car lst) (ru-ent-dxf-code-clear-list (cdr lst) list_dxf_codes
is_stay_value)))
   );_ end of cond
)
```

Листинг 13.138. Функция ru-ent-dxf-code-clear

```
(defun ru-ent-dxf-code-clear (lst dxf_code is_stay_value)
(ru-ent-dxf-code-clear-list lst (list dxf_code) is_stay_value)
)
```

Листинг 13.139. Функция ru-ent-locked

```
(defun ru-ent-locked (ename)
  (ru-layer-is-lock (ru-ent-dxf-code-data 8 (entget ename)))
);_ end of defun
```

Листинг 13.140. Функция ru-ss-get

```
(defun ru-ss-get ()
;|-------------------------------------------------------------------------
 Типовой выбор объектов
Исключаются объекты на блокированных слоях и вне текущего пространства
 Опция "_:L" предотвращает выбор на блокированных слоях
-------------------------------------------------------------------------|;
  (ssget "_:L" (ru-ss-make-filter-space))
);_ end of defun
```

Листинг 13.141. Функция ru-ss-get-first-selection

```
(defun ru-ss-get-first-selection ()
;|-------------------------------------------------------------------------
Возвращает предварительно выбранный набор, если он есть, и PICKFIRST=1, или nil
Пример:(ru-ss-get-first-selection) > <Selection set: 2a>
-------------------------------------------------------------------------|;
(if (and (equal 1 (getvar "PICKFIRST")) (cadr (ssgetfirst)))
    (cadr (ssgetfirst))
 );_ end of if
);_ end of defun
```

Листинг 13.142. Функция ru-ss-get-or-ssfirst

```
(defun ru-ss-get-or-ssfirst (/ selection)
;|-------------------------------------------------------------------------
 Создание набора из предварительно выбранных или из произвольных объектов,
с исключением блокированных
-------------------------------------------------------------------------|;
  (if (not (setq selection (ru-ss-get-first-selection)))
    (setq selection (ru-ss-get))
    (princ
      (ru-string-format "\nПредварительный выбор: %1 объектов"
      (sslength selection))
   );_ end of princ
 );_ end of if
  selection
);_ end of defun
```

Листинг 13.143. Функция ru-ss-join

```
(defun ru-ss-join (ss1 ss2 / ss cnt)
;|-------------------------------------------------------------------------
Объединение наборов
-------------------------------------------------------------------------|;
  (setq ss (ssadd))
  (if (and ss1 ss2)
    (progn
      (setq ss  ss2
            cnt 0
     );_ end of setq
      (repeat (sslength ss1)
        (ssadd (ssname ss1 cnt) ss)
        (setq cnt (1+ cnt))
     );_ end of repeat
   );_ end of progn
 );_ end of if
  (if (and ss1 (not ss2))
    (setq ss ss1)
 );_ end of if
```

```
  (if (and ss2 (not ss1))
    (setq ss ss2)
 );_ end of if
  (if (> (sslength ss) 0)
    (eval ss)
    nil
 );_ end of if
);_ end of defun
```

Листинг 13.144. Функция `ru-ss-make-filter-locked`

```
(defun ru-ss-make-filter-locked (/ result lst)
;|-----------------------------------------------------------------------
 Формирует фильтр на блокированные слои
Пример:
(ru-ss-make-filter-locked)
((-4 . "<NOT")
  (-4 . "<OR")
  (8 . "Конструкции_строительные")
  (8 . "П_Жилые_Коттеджи_ru")
  (8 . "П_Дороги_Авто_ru")
  (-4 . "OR>")
  (-4 . "NOT>")
)
-----------------------------------------------------------------------|;
  (if (setq lst (ru-layer-locked-list))
    (progn
      (setq result '((-4 . "OR>") (-4 . "NOT>")))
      (foreach layer lst
       (setq result (cons (cons 8 layer) result))
     );_ end of foreach
      (setq result (cons '(-4 . "<NOT") (cons '(-4 . "<OR") result)))
   );_ end of progn
 );_ end of if
  result
);_ end of defun
```

Листинг 13.145. Функция `ru-ss-make-filter-space`

```
(defun ru-ss-make-filter-space ()
;|-----------------------------------------------------------------------
Формирование фильтра на текущее пространство
Пример (ru-ss-make-filter-space) > ((-4 . "<NOT") (67 . 0) (-4 . "NOT>"))
-----------------------------------------------------------------------|;
 (if (equal (getvar "CVPORT") 1)
 '((67 . 1))
 '((67 . 0))
);_ end of if
);_ end of defun
```

Листинг 13.146. Функция `ru-ss-remove-locked`

```
(defun ru-ss-remove-locked (selection / start_size end_size i removed)
;|-------------------------------------------------------------------------
 Удаление из набора объектов, расположенных на блокированных слоях
 (ru-ss-remove-locked (ssget))
-------------------------------------------------------------------------|;
  (setq start_size (sslength selection) i 0)
  (while (< i (sslength selection))
    (if (ru-ent-locked (ssname selection i))
      (ssdel (ssname selection i) selection)
      (setq i (1+ i))
   );_ end of if
 );_ end of while
  (setq end_size (sslength selection))
  (if (/= 0 (setq removed (- start_size end_size)))
    (princ (strcat "\nУдалено из набора: "
      (ru-string-number-end removed "объект" "" "а" "ов")
         " на блокированных слоях\n" "\nОсталось "
        (ru-string-number-end end_size "объект" "" "а" "ов") "\n"))
 );_ end of if
  (if (> end_size 0)
    selection
    nil
 );_ end of if
);_ end of defun
```

Листинг 13.147. Функция `ru-get-vla-entsel`

```
(defun ru-get-vla-entsel (message / sel)
;|-------------------------------------------------------------------------
 Выбор с возвратом VLA-объекта
Пример: (ru-get-vla-entsel "Выбери объект")
-------------------------------------------------------------------------|;
  (if (setq sel (ru-get-entsel message))
    (vlax-ename->vla-object (car sel))
 );_ end of if
);_ end of defun
```

Листинг 13.148. Функция `ru-ent-mod`

```
(defun ru-ent-mod (ent value bit / ent_list old_dxf new_dxf)
;|-------------------------------------------------------------------------
Изменение свойства примитива ent, заданного DXF-кодом bit, на значение value
Пример:
Изменить слой последнего примитива на "1"
  (ru-ent-mod (entlast) "1" 8)
Изменить слой последнего примитива на "Бывший_1"
  (ru-ent-mod (entlast) "Бывший_1" 8)
-------------------------------------------------------------------------|;
```

```
 (setq ent_list (entget ent)
      new_dxf  (cons bit
                     (if (and (= bit 62) (= (type value) 'str))
                     (if (= (strcase value) "BYLAYER") 256 0) value)
              );_ end of cons
 );_ end of setq
  (if (/= new_dxf (setq old_dxf (assoc bit ent_list)))
    (progn (entmod (if old_dxf
                     (subst new_dxf old_dxf ent_list)
                     (append ent_list (list new_dxf))
                   );_ end of if
           );_ end of entmod
            (entupd ent)
            (redraw ent)
   );_ end of progn
 );_ end of if
  ent
);_ end of defun
```

Листинг 13.149. Функция `ru-ss-mod`

```
(defun ru-ss-mod (selection value bit / sslen count)
;|-------------------------------------------------------------------------
Изменение у объектов набора selection свойства, заданного DXF-кодом bit, на
значение value
Пример: (ru-ss-mod (ssget) 50 370)
-------------------------------------------------------------------------|;
  (cond ((not selection) nil)
    (t
       (setq sslen (- (sslength selection) 1) count 0)
       (while (<= count sslen)
         (ru-ent-mod (ssname selection count) value bit)
         (setq count (+ 1 count))
      );_ end of while
    count
 )
 );_ end of cond
);_ end of defun
```

Резюме

На настоящем этапе мы имеем базовый набор функций, позволяющих вести дальнейшую разработку. В следующих главах мы разберем создание более сложных функций с использованием дополнительных средств.

ГЛАВА 14

Разработка библиотеки функций с использованием ObjectARX

Библиотеки ObjectARX и среда Microsoft Visual C++ являются самым мощным средством разработки приложений для AutoCAD, причем не только за счет добавления новых команд, но также за счет новых типов примитивов. Как *это* делается, мы описывать не будем. Мы разберемся, для начала, с самым простым использованием ObjectARX для создания дополнительной библиотеки функций, доступных в Visual LISP. Самой известной библиотекой такого класса является DOSLib. Функции библиотеки DOSLib мы также будем использовать в системе ruCAD, но ради любопытства сделаем несколько своих аналогов.

Большим недостатком в создании ARX-библиотек является их привязанность к версии компилятора Visual C++. Если для AutoCAD 2002 нужно было использовать Visual C++ версии 6, то для AutoCAD 2004 и 2005 уже необходимо приобретать Visual Studio .NET с седьмой версией компилятора. Делать это только ради разработки простейших библиотек не имеет смысла. Мы ограничимся разработкой библиотеки для AutoCAD 2002.

Как установить ObjectARX 2002

Пакет ObjectARX SDK можно бесплатно скачать с сайта фирмы Autodesk. Для этого следует заполнить регистрационную форму и получить адрес скачиваемого файла. Скачивать нужно полную версию, размером около 15 Мбайт, включающую справочники и примеры.

После скачивания и инсталляции пакета, например в папку C:\ObjectARX, необходимо инициализировать его в среде MS Visual C++. Для этого следует запустить

```
c:\ObjectArx\utils\ObjARXWiz\WizardSetup.exe
```

После пошагового выполнения этой программы в списке создаваемых проектов Microsoft Visual C++ будет появляться **ObjectARX 2000/2000i/2002 AppWizard** (Мастер разработки приложений ObjectARX 2000/2000i/2002).

Как настроить Microsoft Visual C++ 6.0

После запуска Microsoft Visual C++ необходимо вручную добавить пути поиска заголовочных файлов и библиотек ObjectARX. Для этого из меню **Tools | Options** (Сервис | Настройки) вызовем диалоговое окно настройки, активизируем вкладку **Directories** (Пути), в раскрывающемся списке **Show directories for** (Показать пути для) выберем **Library files** (Файлы библиотек) и к списку **Directories** (Пути) добавим каталог c:\ObjectArx\Lib, а для **Include files**(Заголовочные файлы) — c:\ObjectArx\Inc.

Далее следует установить видимость специальной панели инструментов ObjectARX. Для этого, используя меню **Tools | Customize** (Сервис | Адаптация), вызовем диалоговое окно настройки и на вкладке **Add-ins and Macro Files** (Файлы надстроек и макросов) установим флажок видимости для **ObjectARX 2000/2000i/2002 Add-In** (Надстройка ObjectARX 2000/2000i/2002). Затем на вкладке **Toolbars** (Панели) того же диалогового окна можно изменить название панели по умолчанию (**Toolbar1**) на свое (например, **ObjectARX**).

Теперь все готово для создания библиотек.

Постановка задачи

Для начала мы разработаем небольшую библиотеку с самыми необходимыми функциями, отсутствующими в штатном наборе Visual LISP. Пусть это будут функции чтения и записи строк в INI-файлы, функция запуска приложений в модальном режиме, т. е. с ожиданием завершения работы приложения, и функция вывода стандартного диалога сообщений с возможностью управления его видом. По неведомым причинам таких функций не было в AutoLISP и они не появились в Visual LISP. Возможно, фирма Autodesk считает, что все обязаны сохранять данные только в реестре Windows, как и рекомендует Microsoft, и в файлах конфигурации системы AutoCAD с помощью функций `setcfg` и `getcfg` (этот вопрос мы обсуждали на стадии формирования концепции и решили, что работа с INI-файлами нам понадобится). Итак, нам требуется создать функции, которые из LISP-программ можно вызывать так:

- `(ruarx-set-ini <file_name> <section> <key_name> <value>)` — запись строки `<value>` в переменную `<key_name>` секции `<section>` файла `<file_name>`;
- `(ruarx-get-ini <file_name> <section> <key_name> [<default_value>])` — чтение переменной `<key_name>` секции `<section>` файла `<file_name>` со значением по умолчанию `<default_value>`;
- `(ruarx-run-app <command_line> [<wait>])` — выполнение `<command_line>` (строка запуска программы с возможными параметрами) и ожидание завершения с блокировкой AutoCAD, если `<wait>` равен `T`;
- `(ruarx-win-msg <header> <message> <style>)` — вывод стандартного диалогового окна Windows `MessageBox` с установкой заголовка диалогового окна `<header>`, выводом текста сообщения `<message>` (возможно, в несколько строк), установкой набора кнопок и иконки в зависимости от параметра `<style>`. Диалоговое окно должно возвращать значение, соответствующее кнопке, нажатой при выходе из окна.

Приступаем к реализации задуманного.

Создание заготовки библиотеки с помощью ObjectARX AppWizard

В меню Microsoft Visual C++ выберем пункт **File | New** (Файл | Новый).

В открывшемся диалоговом окне **New** (Новый) (рис. 14.1) на вкладке **Projects** (Проекты) выберем **ObjectARX 2000/2000i/2002 AppWizard**, в поле **Project name** (Имя проекта) впишем имя проекта MainLib, в поле **Location** (Размещение) впишем каталог проекта c:\.ru\cad\source\ObjectARX\MainLib и щелкнем кнопку **OK**.

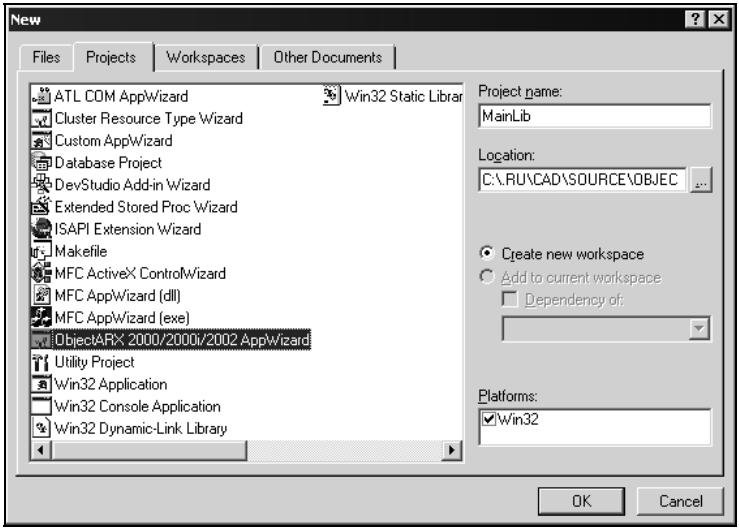


Рис. 14.1. Создание ARX-проекта

В следующем окне **Step 1** (Шаг 1) (рис. 14.2) зададим **Your Registered Developer Symbol** (Ваш зарегистрированный символ разработчика) `ru_` (будем считать это нашим префиксом) и щелкнем кнопку **Next** (Далее).

Замечание

Получить Registered Developer Symbol можно по адресу **http://usa.autodesk.com/adsk/servlet/index?siteID=123112&id=1075006**. Сначала попробуйте вводить по четыре символа, которые вы желаете зарегистрировать. Если они доступны, заполните форму, в которой нужно по возможности честно ответить на ряд хитрых вопросов, и получите зарегистрированный символ. Использование зарегистрированных символов дает гарантию, что функции и команды не будут конфликтовать с командами других разработчиков, также использующих зарегистрированные префиксы. То есть такую же гарантию, как сохранность жизни при соблюдении правил дорожного движения.

На следующем шаге нас любезно проинформируют об основных параметрах проекта (рис. 14.3).

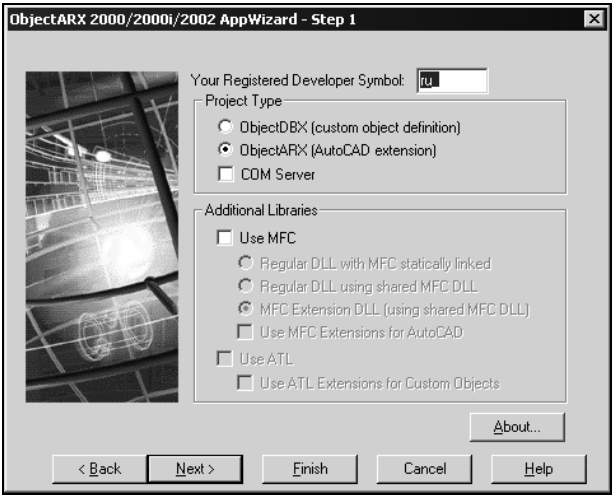


Рис. 14.2. Ввод зарегистрированного символа разработчика

Рис. 14.3. Информация об основных параметрах проекта

Теперь приступим к формированию требуемых функций. Щелкнув по кнопке CMD панели инструментов **ObjectARX**, мы откроем диалоговое окно **ObjectARX Defined Commands** (Команды, созданные в ObjectARX) (рис. 14.4).

Для каждой функции, которую мы хотим включить в библиотеку, выполним следующие операции:

1. Установим переключатель **Register commands via** (Регистрировать команды с помощью) области **New command** (Новые команды) в положение **acedDefun()**.

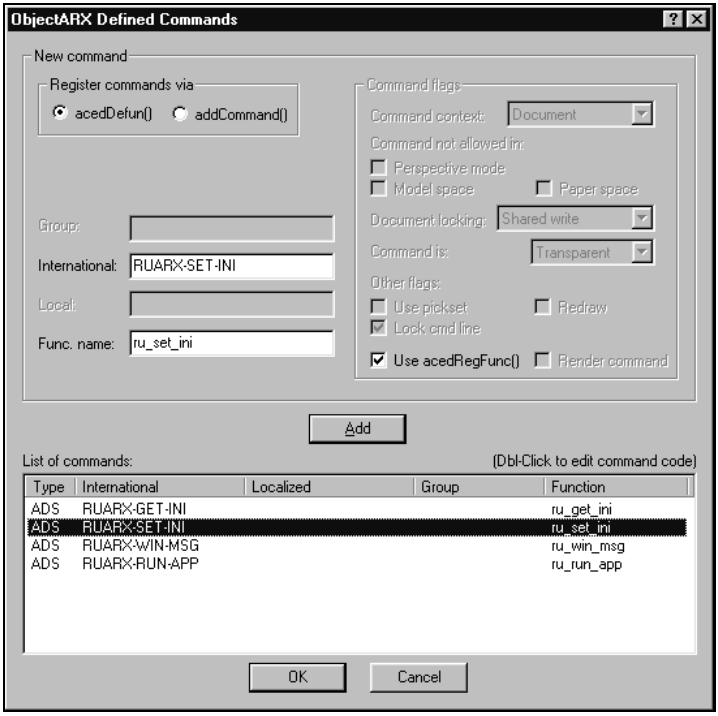


Рис. 14.4. Добавление заготовок функций

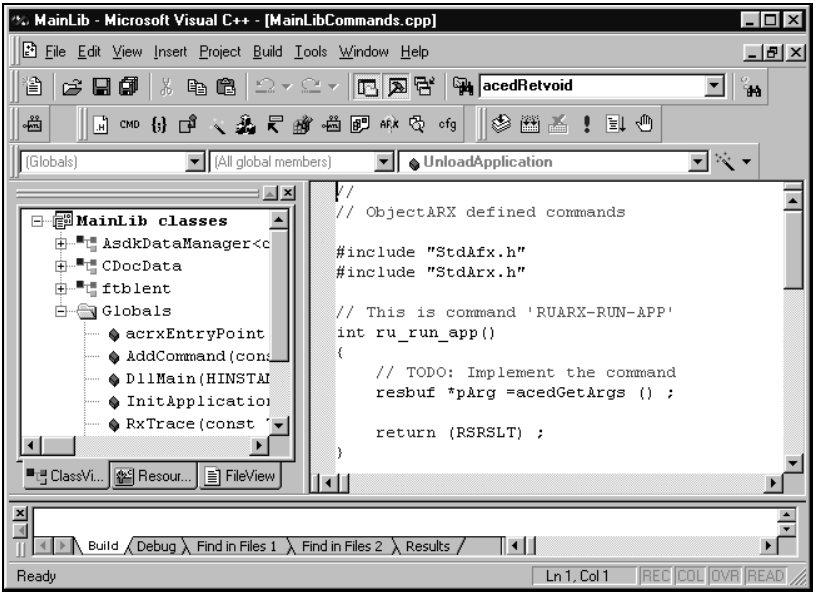


Рис. 14.5. Сгенерированная заготовка проекта

2. Установим флажок **Use acedRegFunc()** (Использовать acedRegFunc ()).
3. В поле **International** (Внешнее имя функции) введем такое имя, под которым мы хотели бы вызывать функцию из LISP.
4. В поле **Func. name** (Внутреннее имя функции) впишем внутреннее имя функции в С-библиотеке.
5. Щелкнем по кнопке **Add** (Добавить).

После занесения всех имен функций и щелчка по кнопке **OK** мы получим заготовку всех файлов проекта, которые останется наполнить содержанием (рис. 14.5).

Анализ полученного кода

Из всего свалившегося на нас богатства нас будут интересовать файлы:

- MainLib.cpp — редактировать ничего не нужно, исследуем только в познавательных целях;
- StdArx.h — в этом файле ничего изменять не нужно, убедимся только, что в нем имеются объявления четырех наших функций:

```
int ru_get_ini();
int ru_set_ini();
int ru_run_app();
int ru_win_msg();
```

- MainLibCommands.cpp — файл, в котором и содержится реализация наших функций.

В MainLibCommands.cpp мы и будем записывать собственный код. Рекомендуется не удалять сгенерированные системой комментарии. Вписываем свои участки кода в отведенные для этого места. В листинге 14.1 все наши добавления по традиции выделяются полужирным шрифтом.

Листинг 14.1. Файл MainLibCommands.cpp

```
//
// ObjectARX defined commands

#include "StdAfx.h"
#include "StdArx.h"

const char* ApplicTitle = "AutoCAD";
const int   MaxLispStrLen = 504;

// This is command 'RU_GET_INI'
int ru_get_ini()
{
// TODO: Implement the command
    resbuf *pArg =acedGetArgs();
    char* ini_file_name;
    char* section_name;
```

```
    char* key_name;
    char default_value[MaxLispStrLen], result_value[MaxLispStrLen];
    char* error_message = NULL;
// Проверка параметров
    //ini_file_name
    if(pArg != NULL){
        if(pArg->restype == RTSTR){
            ini_file_name = pArg->resval.rstring;
            pArg = pArg->rbnext;
        }
        else{
            error_message = "ini_file_name type mismatch";
        }
    }
    else{
        error_message = "too few arguments";
    }

    //section_name
    if(pArg != NULL && error_message == NULL){
        if(pArg->restype == RTSTR){
            section_name = pArg->resval.rstring;
            pArg = pArg->rbnext;
        }
        else{
            error_message = "section_name type mismatch";
        }
    }
    else{
        error_message = "too few arguments";
    }

    //key_name
    if(pArg != NULL && error_message == NULL){
        if(pArg->restype == RTSTR){
            key_name = pArg->resval.rstring;
            pArg = pArg->rbnext;
        }
        else{
            error_message = "key_name type mismatch";
        }
    }
    else{
        error_message = "too few arguments";
    }

    //default_value
    if(pArg != NULL && error_message == NULL){
```

```
        if(pArg->restype == RTSTR){
            strncpy(default_value, pArg->resval.rstring,
                    MaxLispStrLen - 1);
            pArg = pArg->rbnext;
        }
        else{
            error_message = "default_value type mismatch";
        }
    }
    else{
        default_value[0] = '\0'; // пустая строка
    }

    if (pArg != NULL && error_message == NULL){
        error_message = "too many arguments";
    }

    if (error_message != NULL){
        acutPrintf("\nruarx-get-ini: %s",error_message);
        acutPrintf("\nUsage: (ruarx-get-ini ini_file_name
                      section_name key_name [default_value])\n");
        acedRetNil();
        return RSRSLT;
    }

    result_value[0] = '\0';
// Вызов функции WinAPI
    size_t r = GetPrivateProfileString(section_name, key_name,
                    default_value, result_value, MaxLispStrLen - 1,
                    ini_file_name);

    if (strlen(result_value) == r){
        acedRetStr(result_value);
    }
    else{
        acutPrintf("\nruarx-get-ini: error reading INI-file.\n");
        acedRetNil();
    }
   return (RSRSLT);
}//int ru_get_ini()

// This is command 'RU_SET_INI'
int ru_set_ini()
{
// TODO: Implement the command
    resbuf *pArg =acedGetArgs ();

    char* args[4];
    char* error_message = NULL;
```

```
 for (int i = 0; i < 4; i++){
        if (pArg != NULL){
            if (pArg->restype == RTSTR){
                args[i] = pArg->resval.rstring;
                pArg = pArg->rbnext;
            }
            else{
                error_message = "argument's type mismatch";
                break;
            }
        }
        else{
            error_message = "too few arguments";
            break;
        }
    }

    if(pArg != NULL && error_message == NULL){
        error_message = "too many arguments";
    }

    if (error_message != NULL){
        acutPrintf("\nruarx-set-ini: %s",error_message);
        acutPrintf("\nUsage: (ruarx-set-ini ini_file_name section_name
 key_name value)\n");
        acedRetNil();
        return RSRSLT;
    }

    if(WritePrivateProfileString(
        args[1], //section_name,
        args[2], //key_name,
        args[3], //value,
        args[0]  //ini_file_name
       )){
        acedRetT();
    }
    else{
        acedRetNil();
    }
   return (RSRSLT);
}//int ru_set_ini()

// This is command 'RU_RUN_APP'
int ru_run_app()
{
// TODO: Implement the command
    resbuf *pArg =acedGetArgs ();
    char* error_message = NULL;
    char* command_line;
```

```
    //command_line
    if(pArg != NULL){
        if(pArg->restype == RTSTR){
            command_line = pArg->resval.rstring;
            pArg = pArg->rbnext;
        }
        else{
            error_message = "command_line type mismatch";
        }
    }
    else{
        error_message = "command_line is empty";
    }

bool wait_for_exit = false;

    //wait_for_exit
    if(pArg != NULL && error_message == NULL){
        if(pArg->restype == RTT){
            wait_for_exit = true;
            pArg = pArg->rbnext;
        }
    }

    if (pArg != NULL && error_message == NULL){
        error_message = "too many arguments";
    }

    if (error_message != NULL){
        acutPrintf("\nruarx-run-app: %s",error_message);
        acutPrintf("\nUsage: (ruarx-run-app command_line [T])\n");
        acedRetNil();
        return RSRSLT;
    }

    STARTUPINFO cif;
    ZeroMemory(&cif,sizeof(STARTUPINFO));
    PROCESS_INFORMATION pi;
    DWORD exit_code = STILL_ACTIVE;

    if (CreateProcess(NULL, command_line,
        NULL,NULL,FALSE,NULL,NULL,NULL,&cif,&pi) == TRUE){
        if(wait_for_exit){
            HWND acadwnd = adsw_acadMainWnd();
            SetWindowText(acadwnd, "ruCAD - waiting...");
            while(exit_code == STILL_ACTIVE){
                GetExitCodeProcess(pi.hProcess, &exit_code);
            }
```

```
            SetWindowText(acadwnd, ApplicTitle);
            acedRetT();
        }
    }
    else{
        acutPrintf("ruarx-run-app: the program has not been launched");
        acedRetNil();
    }
   return (RSRSLT);
}//int ru_run_app()

// This is command 'RU_WIN_MSG'
int ru_win_msg()
{
// TODO: Implement the command
    resbuf *pArg =acedGetArgs ();

    char* caption;
    char* text;
    UINT style;
    char* error_message = NULL;

    //caption
    if(pArg != NULL){
        if(pArg->restype == RTSTR){
            caption = pArg->resval.rstring;
            pArg = pArg->rbnext;
        }
        else{
            error_message = "caption type mismatch";
        }
    }
    else{
        error_message = "too few arguments";
    }

    //text
    if(pArg != NULL && error_message == NULL){
        if(pArg->restype == RTSTR){
            text = pArg->resval.rstring;
            pArg = pArg->rbnext;
        }
        else{
            error_message = "text type mismatch";
        }
    }
    else{
        error_message = "too few arguments";
    }
```

```
    //style
    if(pArg != NULL && error_message == NULL){
        if(pArg->restype == RTLONG || pArg->restype == RTSHORT){
            if (pArg->restype == RTLONG){
                style = pArg->resval.rlong;
            }
            else{
                style = pArg->resval.rint;
            }
            pArg = pArg->rbnext;
        }
        else{
            error_message = "style type mismatch";
        }
    }
    else{
        style = 0;
    }

    if (pArg != NULL && error_message == NULL){
        error_message = "too many arguments";
    }

    if (error_message != NULL){
        acutPrintf("\nruarx-win-msg: %s"(error_message);
        acutPrintf("\nUsage: (ruarx-win-msg header message style)\n");
        acedRetNil();
        return RSRSLT;
    }

    acedRetInt(MessageBox(adsw_acadMainWnd(), text, caption, style));
   return (RSRSLT);
}//int ru_win_msg()
```

Комментировать написанный нами код нет особого смысла. Большую часть занимают проверки переданных параметров и последующий вызов функций Windows API.

Осталось откомпилировать проект. С помощью пункта меню **Build | Set Active Configuration** (Построение | Установить активную конфигурацию) вызовем диалоговое окно **Set Active Project Configuration** (Установка активной конфигурации проекта) (рис. 14.6) и самоуверенно установим в качестве текущей конфигурации Win32 Release (без отладочных функций).

А теперь нажмем клавишу <F7> (и не забываем отпустить). После некоторого замешательства, вызванного столь "наглыми" действиями, компилятор выдаст:

```
-----Configuration: MainLib - Win32 Release--------
Compiling...
MainLibCommands.cpp
Linking...
   Creating library Release/ru_MainLib.lib and object Release/ru_MainLib.exp
ru_MainLib.arx - 0 error(s), 0 warning(s)
```

А куда же он денется! Полученный файл c:\.ru\cad\Source\ObjectARX\MainLib\Release \ru_MainLib.arx мы переименуем в c:\.ru\cad\Local Settings\Application Data\ruCAD \AutoCAD\15\ru_MainLib.arx.

Увы, использовать его мы сможем только в системе AutoCAD 2002. Для AutoCAD 2004, как минимум, необходимо перекомпилировать проект в Visual C++ версии 7.

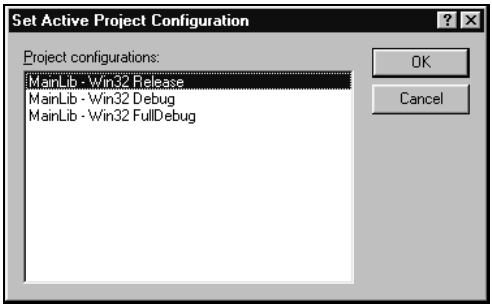


Рис. 14.6. Диалоговое окно **Set Active Project Configuration**

Как устроен ARX

Если уж мы зачем-то создали ARX-приложение, то давайте хоть разберемся, как оно работает. Для этого нам придется заглянуть в другие файлы, в которые мы ничего не записывали. Вообще-то любой ARX-файл — это динамически загружаемая библиотека (DLL), только с расширением arx.

Заглянув в файл определений MainLib.def (листинг 14.2), мы увидим, что экспортируются две функции.

Листинг 14.2. Файл MainLib.def

```
LIBRARY      "ru_MainLib"
EXPORTS
   acrxEntryPoint      PRIVATE
   acrxGetApiVersion   PRIVATE
```

При честном подходе со стороны фирмы Autodesk такую библиотеку можно было бы написать в любой среде программирования, и тем более, без привязки к версии компилятора. Увы! Все "подлости" таятся в функции `acrxGetApiVersion`. Теоретически она выдает только номер версии, практически же делает что-то еще, не позволяя загрузить "неправильный" ARX в AutoCAD.

Как работает функция *acrxEntryPoint*

Функция `acrxEntryPoint` (листинг 14.3) является главной. В ней и происходит вся работа.

Листинг 14.3. Фрагмент файла MainLib.cpp. Функция `acrxEntryPoint`

```
extern "C" AcRx::AppRetCode
acrxEntryPoint(AcRx::AppMsgCode msg, void* pkt)
{
  switch (msg) {
  case AcRx::kInvkSubrMsg:
    dofun(); // Выполнение функции при вызове
    break;
  case AcRx::kUnloadDwgMsg:
    funcunload(); // Выгрузка функций при выгрузке DWG
    break;
  case AcRx::kLoadDwgMsg:
    funcload(); // Загрузка функций
    break;
  case AcRx::kInitAppMsg:
    // Comment out the following line if your
    // application should be locked into memory
    acrxDynamicLinker->unlockApplication(pkt);
    acrxDynamicLinker->registerAppMDIAware(pkt);
    InitApplication();
    break;
  case AcRx::kUnloadAppMsg:
    UnloadApplication(); // Выгрузка всего приложения
    break;
}
return AcRx::kRetOK;
}
```

Эта функция обрабатывает поступающие от системы AutoCAD сообщения и выполняет соответствующие действия.

Как регистрируются функции для Visual LISP

Регистрация пользовательских функций, которые будут доступны в Visual LISP, осуществляется функцией `acedDefun` во время выполнения функции `funcload` при событии `kLoadDwgMsg` (листинг 14.4).

Листинг 14.4. Фрагмент файла MainLib.cpp. Функция `funcload`

```
static int funcload(void)
{
  int i;
  for (i = 1; i < ELEMENTS(exfun); i++) {
    if (!acedDefun(exfun[i].name, i))
      return RTERROR;
    if (exfun[i].regFunc)
      acedRegFunc(exfun[i].fptr, i);
  }
  return RTNORM;
}
```

Массив загружаемых функций определен в таблице AFX_ADS_FUNC_TABLE, выделенной редактором Visual C серым цветом (листинг 14.5). При извещении о событии kUnloadDwgMsg эти функции будут выгружены.

Листинг 14.5. Фрагмент файла MainLib.cpp. Массив загружаемых функций

```
//{{AFX_ADS_FUNC_TABLE
typedef struct {
    char    *name;
    int     (*fptr)();
    BOOL    regFunc;
    BOOL    renderCmd;
} ftblent;

#define ELEMENTS(array) (sizeof(array)/sizeof((array)[0]))

ftblent exfun[] = {
    {"", NULL, FALSE, FALSE },
    {"ruarx-get-ini", ru_get_ini, TRUE, FALSE },
    {"ruarx-set-ini", ru_set_ini, TRUE, FALSE },
    {"ruarx-run-app", ru_run_app, TRUE, FALSE },
    {"ruarx-win-msg", ru_win_msg, TRUE, FALSE },
}; //}}AFX_ADS_FUNC_TABLE
```

При получении извещения о событии kInvkSubrMsg (вызов функции) вызываемая функция выполняется посредством функции dofun (листинг 14.6).

Листинг 14.6. Фрагмент файла MainLib.cpp. Функция dofun

```
static int dofun(void)
{
  int val,
      rc;

  acedRetVoid();

  if ((val = acedGetFunCode()) < 1 || val > ELEMENTS(exfun))
    return RTERROR;
#ifdef RENDER
  if (exfun[val].renderCmd)
    if (!InitRender(false))
      return RTERROR;
#endif
  rc = (*exfun[val].fptr)();
  return ((rc == RTNORM) ? RSRSLT:RSERR);
}
```

Как используется буфер результатов

Буфер результатов (resbuf) используется для обмена данными между функциями AutoLISP и ObjectARX. Это структура (листинг 14.7), которая оперирует всеми типами данных AutoCAD.

Листинг 14.7. Структура `resbuf`

```
union ads_u_val {
       ads_real rreal;
       ads_real rpoint[3];
       short rint;
       char *rstring;
       long rlname[2];
       long rlong;
       struct ads_binary rbinary;
};
struct resbuf {
     struct resbuf *rbnext;
     short restype;
     union ads_u_val resval;
};
```

В одной структуре `resbuf` может быть указатель на другую вложенную структуру `resbuf`. Таким образом можно передать несколько аргументов. Поле `restype` содержит целочисленный код, указывающий, какой тип данных загружается в объединение[1] `resval`.

Разберем для примера функцию `ru_get_ini`. В LISP она вызывается так:

```
(ruarx-get-ini <ini_file_name> <section_name> <key_name> [<default_value>]))
```

Функции передается не менее трех параметров строкового типа. В ARX сначала формируется указатель на массив аргументов:

```
resbuf *pArg =acedGetArgs();
```

Далее проверяется тип каждого аргумента, например:

```
if(pArg->restype == RTSTR)
```

и формируется указатель текущего аргумента с переводом указателя на следующий аргумент:

```
ini_file_name = pArg->resval.rstring;
pArg = pArg->rbnext;
```

Или выдается сообщение об ошибке, а функция возвращает `nil` посредством

```
acedRetNil();
```

Различные коды возврата определены в файле adscodes.h. Например, у нас использованы:

- `RSRSLT` 1 — возврат результата;
- `RSERR` 3 — ошибка вычисления, нет результата;
- `RTNONE` 5000 — нет результата;
- `RTREAL` 5001 — вещественное число;

[1] Объединение (`union`) — тип данных.

- RTSHORT 5003 — короткое целое;
- RTSTR 5005 — строка;
- RTLONG 5010 — длинное целое;
- RTNIL 5019 — nil;
- RTT 5021 — T;
- RTRESBUF 5023 — resbuf;
- RTNORM 5100 — успешное выполнение.

Мы намеренно дали очень краткие пояснения. Программистам, работающим в C++, все и так понятно, а попытки популярно, в несколько строк, объяснить суть происходящего ни к чему хорошему не приведут.

Работа с ARX-функциями в Visual LISP

Конечной целью разработки этой библиотеки является использование дополнительных функций в Visual LISP. Полученные функции мы можем вызывать напрямую, но все не так просто.

Во-первых, ARX разработан только для системы AutoCAD 2002. Нужно предусмотреть вариант работы и в AutoCAD 2004.

Во-вторых, функция (ruarx-win-msg <header> <message> <style>), выводящая стандартное диалоговое окно, довольно сложна в использовании. В исходном тексте вы

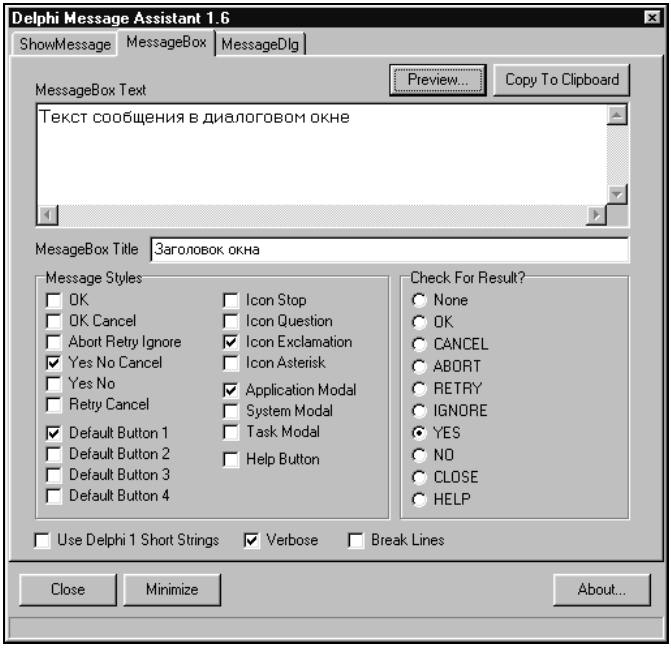


Рис. 14.7. Установка параметров диалогового окна

видели, что мы вызвали функцию Windows API `MessageBox`. Именно эта функция выводит различные простые диалоговые окна, с которыми мы постоянно сталкиваемся в Windows. Вид диалогового окна определяется параметром `<style>` и вся хитрость заключается в правильном задании этого простенького целочисленного параметра. Чтобы проиллюстрировать, какое сочетание параметров требуется учитывать, приведем скриншот одного из многочисленных визуальных конструкторов диалогового окна в Delphi (рис. 14.7).

При показанных на рис. 14.7 установках вызов функции на Delphi будет иметь вид

```
if Application.MessageBox('Текст сообщения в диалоговом окне',
                          'Заголовок окна',
                            MB_YESNOCANCEL
                           +MB_ICONEXCLAMATION
                           +MB_DEFBUTTON1
                           +MB_APPLMODAL) = ID_YES then
     begin
     end;
```

а диалоговое окно во время выполнения будет выглядеть, как показано на рис. 14.8.

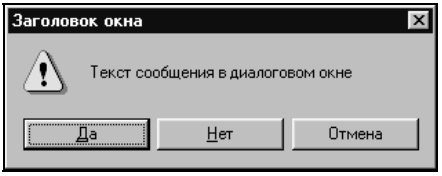


Рис. 14.8. Диалоговое окно функции `MessageBox`

Именованные константы `MB_YESNOCANCEL`, `MB_ICONEXCLAMATION` и `MB_DEFBUTTON1`, сложением которых определяется стиль диалогового окна, в Delphi определены в модуле Windows.pas:

```
MB_OK                       = $00000000;
MB_OKCANCEL                 = $00000001;
MB_ABORTRETRYIGNORE         = $00000002;
MB_YESNOCANCEL              = $00000003;
MB_YESNO                    = $00000004;
MB_RETRYCANCEL              = $00000005;

MB_ICONHAND                 = $00000010;
MB_ICONQUESTION             = $00000020;
MB_ICONEXCLAMATION          = $00000030;
MB_ICONASTERISK             = $00000040;
MB_USERICON                 = $00000080;
MB_ICONWARNING              = MB_ICONEXCLAMATION;
MB_ICONERROR                = MB_ICONHAND;
MB_ICONINFORMATION          = MB_ICONASTERISK;
MB_ICONSTOP                 = MB_ICONHAND;

MB_DEFBUTTON1               = $00000000;
MB_DEFBUTTON2               = $00000100;
```

```
MB_DEFBUTTON3                   = $00000200;
MB_DEFBUTTON4                   = $00000300;

MB_HELP                         = $00004000;
```

Для Visual C++ в файле winuser.h соответствующие константы объявлены так:

```
#define MB_OK                       0x00000000L
#define MB_OKCANCEL                 0x00000001L
#define MB_ABORTRETRYIGNORE         0x00000002L
#define MB_YESNOCANCEL              0x00000003L
#define MB_YESNO                    0x00000004L
#define MB_RETRYCANCEL              0x00000005L

#define MB_ICONHAND                 0x00000010L
#define MB_ICONQUESTION             0x00000020L
#define MB_ICONEXCLAMATION          0x00000030L
#define MB_ICONASTERISK             0x00000040L

#define MB_USERICON                 0x00000080L
#define MB_ICONWARNING              MB_ICONEXCLAMATION
#define MB_ICONERROR                MB_ICONHAND

#define MB_ICONINFORMATION          MB_ICONASTERISK
#define MB_ICONSTOP                 MB_ICONHAND

#define MB_DEFBUTTON1               0x00000000L
#define MB_DEFBUTTON2               0x00000100L
#define MB_DEFBUTTON3               0x00000200L
#define MB_HELP                     0x00004000L
```

В Visual LISP также имеются аналогичные именованные константы для установки свойств диалоговых окон. Определены они так:

```
;;; Коды кнопок
(setq VLAX-VBOK          1)
(setq VLAX-VBCANCEL      2)
(setq VLAX-VBABORT       3)
(setq VLAX-VBRETRY       4)
(setq VLAX-VBIGNORE      5)
(setq VLAX-VBYES         6)
(setq VLAX-VBNO          7)
;;; Сочетания кнопок
(setq VLAX-VBOKONLY              0)
(setq VLAX-VBOKCANCEL            1)
(setq VLAX-VBABORTRETRYIGNORE    2)
(setq VLAX-VBYESNOCANCEL         3)
(setq VLAX-VBYESNO               4)
(setq VLAX-VBRETRYCANCEL         5)
;;; Кнопка по умолчанию
(setq VLAX-VBDEFAULTBUTTON1        0)
(setq VLAX-VBDEFAULTBUTTON2      256)
(setq VLAX-VBDEFAULTBUTTON3      512)
```

```
;;; Обозначения значков
(setq VLAX-VBQUESTION           32)
(setq VLAX-VBEXCLAMATION        48)
(setq VLAX-VBINFORMATION        64)
```

Именованные константы в разных системах программирования введены с одинаковой целью — предоставить программисту осмысленные наименования загадочных чисел, имеющих различное представление (`$00000004`, `0x00000004L` или просто `4`), однако механизм их использования разный. В компилирующих системах Delphi и C++ значения констант подставляются компиляторами во время трансляции исходного текста в соответствующий код. В интерпретирующей системе Visual LISP именованные константы фактически являются глобальными переменными. Программист может их нечаянно изменить или включить в список локальных переменных (этому поспособствует анализатор кода). Не все константы определены в Visual LISP[1] — нет, например, подразумевающейся (`setq VLAX-VBSTOP 16`). В библиотеке Express Tools, использующей собственные диалоговые окна, также применены аналогичные константы, например (`setq ACET:OKCANCEL 1`).

Мы дополнительных глобальных переменных вводить не будем и при разработке функции работы с диалоговыми окнами (см. листинг 14.11) воспользуемся именованными константами Visual LISP.

Варианты работы с INI-файлами

Для решения проблемы AutoCAD 2004 мы сделаем специальную функцию-оболочку.

Напомним, что в файле acaddoc.lsp имеются строки

```
(if (< (ru-acad-ver) 16)
  (progn
    (princ "\nЗагружаю MainLib...\n")
    (if (not (member "ru_MainLib.arx" (arx)))
      (arxload (findfile "ru_MainLib.arx"))
    );_ end of if
    (princ "\n......загружена...\n")
    (setq *ru_use_ru_arx* t)
  );_ end of progn
  (progn
    (princ "\nB AutoCAD 2004 работаем с DOSLib 2004\n")
    (setq *ru_use_ru_arx* NIL)
  )
);_ end of if
(if (or (ru-doslib-load) *ru_use_ru_arx*)
  (ru-init-start-rucad)
  (alert (strcat "Система не сможет работать. "
       "Не загружена ни MainLib, ни DOSLib"))
)
```

Как видите, библиотеку DOSLib мы обязательно загружаем, и без ее загрузки система ruCAD вообще откажется работать. Так почему-же мы просто не используем ее

[1] Подобные недостатки замечаются и в отношении некоторых других констант.

функции всегда? Зачем вообще начали разрабатывать свою ARX-библиотеку, дублирующую функции DOSLib?

Откроем возмущенным читателям секрет: мы вообще могли бы ни использовать DOSLib, ни создавать свою библиотеку. Все требуемые функции можно разработать с применением COM-технологий. Это будет даже лучше, т. к. при использовании COM мы уже совершенно не будем зависеть ни от версий AutoCAD, ни от версий Visual C++, ни от всяких "сговоров" фирм Autodesk и Microsoft. Более того, если функции из ARX-библиотек можно применять только в системе AutoCAD, то COM позволяет использовать объекты в любых приложения. А такую мелочь, как чтение-запись строк в INI-файлы, мы делали еще на языке AutoLISP в системе AutoCAD R10.

Преимущество ARX-функций в скорости работы, но для пользователя не имеет значения, читается ли переменная за сотую или тысячную долю секунды, ведь в программе количество таких операций исчисляется максимум десятками.

В данном случае ARX-библиотеку мы разработали "не корысти ради, а токмо волею пославшей мя супруги", т. е. в учебных целях.

Итак, напишем функции-оболочки для работы с INI-файлами (листинги 14.8 и 14.9).

Листинг 14.8. Функция `ru-ini-read`

```
(defun ru-ini-read (ini_name section var default)
;; Делаем единственное место для замены функции нашего ARX
;; на функции из DOSLib
    (if *ru_use_ru_arx*
      (ruarx-get-ini ini_name section var default)
      (dos_getini section var ini_name default)
   );_ end of if
);_ end of defun
```

Именно эту функцию мы будем вызывать во всех других функциях и программах. При необходимости мы сможем и ее переопределить в единственном месте, и, например, заменить ARX-функции на чтение данных на LISP:

```
(defun ru-ini-read (ini_name section var default)
  (ru-ini-read-direct ini_name section var default)
);_ end of defun
```

Листинг 14.9. Функция `ru-ini-write`

```
(defun ru-ini-write (ini_name section var string)
  (if *ru_use_ru_arx*
    (ruarx-set-ini ini_name section var string)
    (dos_setini section var string ini_name)
 );_ end of if
);_ end of defun
```

Запуск приложений

По изложенным ранее причинам делаем функцию-оболочку и для запуска приложений (листинг 14.10).

Листинг 14.10. Функция ru-app-run

```
(defun ru-app-run (command_line wait)
(if *ru_use_ru_arx*
   (ruarx-run-app command_line wait)
   (if wait
      (dos_exewait command_line 0)
      (dos_execute command_line 0)
  );_ end of if
);_ end of if
);_ end of defun
```

Создание безопасной оболочки для окна сообщений

А теперь напишем дополнительную функцию, скрывающую низкоуровневые сложные вызовы (листинг 14.11).

Листинг 14.11. Функция ru-msg-box-ex

```
(defun ru-msg-box-ex (header text buttons_set_name icon_name
 default_button_number /
win_btn_number win_def_btn_number win_icon_number result)
;|-----------------------------------------------------
Аргументы:
header - заголовок окна
text - текст сообщения с разделением на строки по \n
buttons_set_name - символьное обозначение набора кнопок
   Допустимые наборы кнопок buttons_set_name:
  "OK"                  Только кнопка OK
  "OK_CANCEL"           Кнопки OK Отмена
  "RETRY_CANCEL"        Кнопки Повтор Отмена
  "YES_NO"              Кнопки Да Нет
  "YES_NO_CANCEL"       Кнопки Да Нет Отмена
  "ABORT_RETRY_IGNORE"  Кнопки Стоп Повтор Пропустить
icon_name - символьное обозначение иконки
   Допустимые обозначения иконок:
        "ICONEXCLAMATION"
        "ICONINFORMATION"
        "ICONQUESTION"
        "ICONSTOP"
default_button_number - номер кнопки по умолчанию (1, 2 или 3)

Функция возвращает одно из значений или NIL:
Возможные результаты:
 "ABORT" - нажата кнопка Abort (Прервать)
 "CANCEL" - нажата кнопка Cancel (Отмена)
```

```
 "IGNORE" - нажата кнопка Ignore (Пропустить)
 "NO" - нажата кнопка No (Нет)
 "OK" - нажата кнопка OK
 "RETRY" - нажата кнопка Retry (Повтор)
 "YES" - нажата кнопка Yes (Да)
-------------------------------------------------------|;
  (cond
    ((= buttons_set_name "OK")
     (setq win_btn_number vlax-vbokonly);; 0
   )
    ((= buttons_set_name "OK_CANCEL")
     (setq win_btn_number vlax-vbokcancel);; 1
   )
    ((= buttons_set_name "ABORT_RETRY_IGNORE")
     (setq win_btn_number vlax-vbabortretryignore);; 2
   )
    ((= buttons_set_name "YES_NO_CANCEL")
     (setq win_btn_number vlax-vbabortretryignore);; 3
   )
    ((= buttons_set_name "YES_NO")
     (setq win_btn_number vlax-vbyesno);; 4
   )
    ((= buttons_set_name "RETRY_CANCEL")
     (setq win_btn_number vlax-vbretrycancel);; 5
   )
    (t
     (setq win_btn_number vlax-vbokonly);; 0
   )
 );_ end of cond
  (cond
    ((= icon_name "ICONSTOP")
     (setq win_icon_number 16);; константы нет
   )
    ((= icon_name "ICONQUESTION")
     (setq win_icon_number vlax-vbquestion);; 32
   )
    ((= icon_name "ICONEXCLAMATION")
     (setq win_icon_number vlax-vbexclamation);; 48
   )
    ((= icon_name "ICONINFORMATION")
     (setq win_icon_number vlax-vbinformation);; 64
   )
    (t
     (setq win_icon_number 0)
   )
 );_ end of cond
(cond
    ((= default_button_number 1)
     (setq win_def_btn_number vlax-vbdefaultbutton1);; 0
   )
```

```
    ((= default_button_number 2)
     (setq win_def_btn_number vlax-vbdefaultbutton2);; 256
    )
     ((= default_button_number 3)
      (setq win_def_btn_number vlax-vbdefaultbutton3);; 512
    )
    ;| Есть еще 4-я кнопка, не имеющая константы VLAX-VBDEFAULTBUTTON4, но мы ее
не будем применять
     ((= default_button_number 4)
      (setq win_def_btn_number 768)
    )
     |;
     (T
      (setq win_def_btn_number vlax-vbdefaultbutton1)
    )
 );_ end of cond

  (setq result
         (ruarx-win-msg
           header
           text
            (logior
                  win_btn_number
                  win_icon_number
                  win_def_btn_number
           );_ end of logior
         );_ end of ru-WinMsg
 );_ end of print
  (cond
;;; результаты
     ((= result vlax-vbok);; 1
      "OK"
    )
     ((= result vlax-vbcancel);; 2
      "CANCEL"
    )
     ((= result vlax-vbabort);; 3
      "ABORT"
    )
     ((= result vlax-vbretry);; 4
      "RETRY"
    )
     ((= result vlax-vbignore);; 5
      "IGNORE"
    )
     ((= result vlax-vbyes);; 6
      "YES"
    )
     ((= result vlax-vbno);; 7
      "NO"
    )
```

```
    ((= result 8)
    ;; Соответствующая константа VLAX-VBCLOSE не определена
     "CLOSE"
   )
    ((= result 9)
    ;; Соответствующая константа VLAX-VBHELP не определена
     "HELP"
   )
    (t
     nil
   )
 );_ end of cond
);_ end of defun
```

Приведенная функция "оборачивает" ARX-функцию и фактически предоставляет программисту более удобный, "человеческий" интерфейс. Результаты возвращаются в виде строк для удобства восприятия. Но и эта функция достаточно сложна и напрямую будет вызываться в уникальных случаях. Для массового применения мы напишем несколько простых высокоуровневых функций. Но прежде вновь решим проблему с AutoCAD 2004 (листинг 14.12).

Листинг 14.12. Функция `ru-msg-messagebox`

```
(defun ru-msg-messagebox (header text buttons_set_name icon_name
                          default_button_number)
    (if *ru_use_ru_arx*
    (ru-msg-box-ex header text buttons_set_name icon_name
                                    default_button_number)
    (ru-msg-dos-msgboxex header text buttons_set_name icon_name 1)
   );_ end of if
);_ end of defun
```

Эта функция является унифицированным интерфейсом для вызова, в зависимости от значения глобальной переменной `*ru_use_ru_arx*`, диалоговых окон из библиотек ru_MainLib.arx или DosLib2004.arx. Но т. к. функции библиотеки DOSLib имеют другие аргументы, напишем еще одну оболочку, уже для DOSLib (листинг 14.13).

Листинг 14.13. Функция `ru-msg-dos-msgboxex`

```
(defun ru-msg-dos-msgboxex (header text buttons_set_name icon_name
default_button_number / list_result buttons_list icon result)

;;; Аргументы и результаты такие же, как и в функции ru-msg-box-ex

  (cond
    ((= buttons_set_name "OK_CANCEL")
    ;; для оценки результатов
     (setq buttons_list (list "OK" "Отменить")
           list_result  (list "OK" "CANCEL")
    );_ end of setq
   )
```

```
    ((= buttons_set_name "ABORT_RETRY_IGNORE")
     (setq buttons_list (list "Прервать" "Повторить" "Пропустить")
           list_result  (list "ABORT" "RETRY" "IGNORE")
    );_ end of setq
   )
    ((= buttons_set_name "YES_NO_CANCEL")
     (setq buttons_list (list "Да" "Нет" "Отменить")
           list_result  (list "YES" "NO" "CANCEL")
    );_ end of setq
   )
    ((= buttons_set_name "YES_NO")
     (setq buttons_list (list "Да" "Нет")
           list_result  (list "YES" "NO")
    );_ end of setq
   )
    ((= buttons_set_name "NO_YES")
     (setq buttons_list (list "Нет" "Да")
           list_result  (list "NO" "YES")
    );_ end of setq
   )
    ((= buttons_set_name "RETRY_CANCEL")
     (setq buttons_list (list "Повторить" "Отменить")
           list_result  (list "RETRY" "CANCEL")
    );_ end of setq
   )
    (t
     (setq buttons_list (list "OK")
           list_result  (list "OK")
    );_ end of setq
   )
);_ end of cond
 (cond
   ((= icon_name "ICONSTOP")
    (setq icon 5)
   )
   ((= icon_name "ICONQUESTION")
    (setq icon 4)
   )
   ((= icon_name "ICONEXCLAMATION")
    (setq icon 1)
   )
   ((= icon_name "ICONINFORMATION")
    (setq icon 3)
   )
   (T
    (setq icon 2)
   )
);_ end of cond
 (if (setq result (dos_msgboxex text header buttons_list icon))
   (nth result list_result)
```

```
    nil
 );_ end of if
);_ end of defun
```

В этой функции мы просто преобразовали аргументы и результаты в формат функции `dos_msgboxex`.

Разработка группы функций для вывода диалоговых окон

Осталось самое легкое — написать несколько функций (листинги 14.14—14.18), которые мы чаще всего и будем применять, не задумываясь о всяких сложностях с передачей параметров. В большинстве программ сложного и не требуется, нужны простые окна сообщений (см. рис. 14.9—14.13), диалоги типа Да/Нет и т. п. Начнем с замены стандартной функции `alert` (листинг 14.14).

Листинг 14.14. Функция `ru-msg-alert`

```
(defun ru-msg-alert (text)
;;; выводит сообщение в несколько строк по признаку "\n"
  (if (not (equal 4 (logand 4 (getvar "cmdactive"))))
    (if (null (ru-msg-messagebox
                (strcat (ru-user-long-name) "!")
                text  "OK" "ICONSTOP" 1
              );_ end of ru-msg-messagebox
       );_ end of null
      (princ
        (strcat
          "\nНЕ МОГУ ПОКАЗАТЬ ДИАЛОГ, ВЫВОЖУ СООБЩЕНИЕ В КОМАНДНУЮ СТРОКУ:\n"
          text
       );_ end of strcat
     );_ end of princ
   );_ end of if
    (princ (strcat "\n" text))
 );_ end of if
);_ end of defun
```

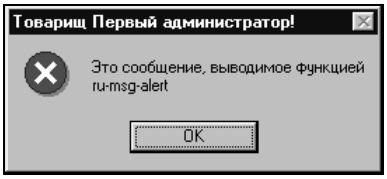


Рис. 14.9. Диалоговое окно, выведенное функцией `ru-msg-alert`

Как видите, даже в таком простом случае мы предусмотрели маловероятный вариант, при котором диалоговое окно по каким-то форс-мажорным обстоятельствам не

может быть выведено. В этом случае сообщение будет напечатано в командной строке. Заголовок окна мы создаем функцией `ru-user-long-name`, которая выводит "человеческое" обращение к конкретному пользователю, типа "Разлюбезная Екатерина Матвеевна" или "Уважаемый Николай Николаевич" — в зависимости от предпочтений пользователя.

Листинг 14.15. Функция `ru-msg-info`

```
(defun ru-msg-info (text)
;;; Выводит сообщение в несколько строк по признаку "\n"
  (if (null (ru-msg-messagebox
              (strcat (ru-user-long-name) "! ")
              text "OK" "ICONEXCLAMATION" 1
            );_ end of ru-msg-messagebox
      );_ end of null
    (princ
      (strcat
        "\nНЕ МОГУ ПОКАЗАТЬ ДИАЛОГ, ВЫВОЖУ СООБЩЕНИЕ В КОМАНДНУЮ СТРОКУ:\n"
        text
      );_ end of strcat
    );_ end of princ
 );_ end of if
);_ end of defun
```

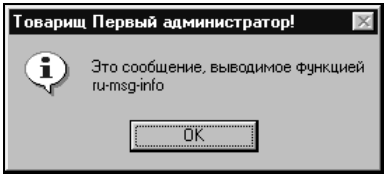


Рис. 14.10. Диалоговое окно, выведенное функцией `ru-msg-info`

Листинг 14.16. Функция `ru-dlg-yes-no-cancel`

```
(defun ru-dlg-yes-no-cancel (text / result)
;;; Выводит сообщение в несколько строк по признаку "\n"
;;; ответ "YES"|"NO"|"CANCEL"
  (if (setq result (ru-msg-messagebox
                     (strcat (ru-user-long-name)
                             ", прошу ПОДУМАТЬ и ответить"
                     );_ end of strcat
                     (strcat text "?") "YES_NO_CANCEL" "ICONQUESTION" 1
                   );_ end of ru-msg-messagebox
      );_ end of setq
    result
    (progn
      (princ
        (strcat
          "\nОШИБКА! Не выведен диалог Да_Нет_Отмена, вывожу простой вопрос!"
```

```
      );_ end of strcat
    );_ end of princ
     (if (ru-dlg-yes-cml text)
       "YES"
       "NO"
    );_ end of if
  );_ end of progn
 );_ end of if
);_ end of defun
```

Рис. 14.11. Диалоговое окно, выведенное функцией `ru-dlg-yes-no-cancel`

Листинг 14.17. Функция `ru-yes`

```
(defun ru-yes (text / result)
;;; Возвращает T или NIL
  (setq result
        (ru-msg-messagebox
          (strcat (ru-user-long-name) ", прошу ответить")
          (strcat text "?")
          "YES_NO" "ICONQUESTION" 1
        );_ end of ru-msg-messagebox
 );_ end of setq
  (if (null result)
    (progn
      (princ
        "\nНЕ МОГУ ПОКАЗАТЬ ДИАЛОГ, ВЫВОЖУ ЗАПРОС В КОМАНДНУЮ СТРОКУ:\n"
     );_ end of princ
      (ru-dlg-yes-cml text)
   );_ end of progn
    (= result "YES")
 );_ end of if
);_ end of defun
```

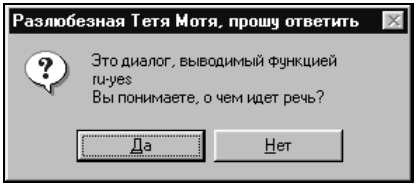


Рис. 14.12. Диалоговое окно, выведенное функцией `ru-yes`

Листинг 14.18. Функция `ru-no`

```
(defun ru-no (text / result)
  (setq result (ru-msg-messagebox
                 (strcat (ru-user-long-name)
                         ", прошу ПОДУМАТЬ и ответить"
                );_ end of strcat
                 (strcat text "?")
                 "YES_NO" "ICONSTOP" 2
               );_ end of ru-msg-messagebox
 );_ end of setq
  (if (null result)
    (progn
      (princ
        "\nНЕ МОГУ ПОКАЗАТЬ ДИАЛОГ, ВЫВОЖУ ЗАПРОС В КОМАНДНУЮ СТРОКУ:\n"
     );_ end of princ
      (ru-dlg-no-cml text)
   );_ end of progn
    (= result "NO")
 );_ end of if
);_ end of defun
```

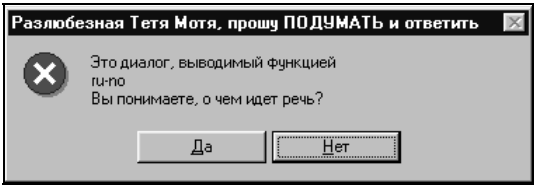


Рис. 14.13. Диалоговое окно, выведенное функцией `ru-no`

Любителям простоты

Наверняка многие читатели возмущены, что мы так сложно решаем такие простые вопросы. Уж они-то знают, как просто это делается! Знаем, конечно, и мы. Можно, например, написать функцию, выполняющую выражение VBA (листинг 14.19).

Листинг 14.19. Функция `ru-msg-vla-box`

```
(defun ru-msg-vla-box (title msg)
 (vla-eval (ru-obj-get-acad-object)
   (strcat "MsgBox  \"" msg "\"" ", " "vbInformation" ", " "\"" title "\"")
);_ end of vla-eval
);_ end of defun
```

Эта функция, в конечном итоге, вызовет ту же функцию Windows API, только через несколько посредников.

Возможны и другие варианты, такие как конструирование собственных форм. Но это уже темы других глав.

ГЛАВА 15

Разработка библиотечных функций с использованием Delphi

В *главе 14* мы разработали несколько простых библиотечных функций с использованием ObjectARX и системы разработки Visual C++. При этом мы в общем-то и не обращались к специальным возможностям ObjectARX. Такие функции можно разработать в любой среде программирования, но для того, чтобы они стали доступными для применения в Visual LISP, нам и понадобился пакет ObjectARX. Именно так в недалеком прошлом создавались расширения для AutoLISP, которые можно было пересчитать по пальцам. Мы столкнулись и с неприятной необходимостью использования для очередной версии системы AutoCAD очередной версии Visual C++. К счастью, фирма Autodesk компенсировала моральный ущерб, внедрив в систему AutoCAD технологию ActiveX Automation, которую мы уже расхваливали в *главе 9*. Там же мы наметили основные направления использования в нашей системе среды быстрой разработки приложений Borland Delphi. В этой главе мы займемся реализацией наших "коварных" планов. Среда разработки Borland Delphi весьма популярна на территории бывшего СССР по целому ряду причин. Delphi с успехом используют многие разработчики программ из той категории читателей, на которых рассчитана наша книга — не "настоящих" программистов, а инженеров-прикладников, занимающихся разработкой САПР (об этом мы упоминали *во введении*). Все приводящиеся в этой и последующей главах примеры рассчитаны именно на таких специалистов.

К сожалению, очень сложно описать в этой книге, в условиях дефицита места и времени, целый ряд довольно сложных тем (COM-технологии, работа с XML и базами данных и т. п.). Обычно каждая такая тема раскрывается, да и то обзорно, в отдельной книге, потолще чем наша. Соревноваться с авторами фундаментальных изданий мы не можем, поэтому объяснения будут самыми минимальными. При разработке приложений нам придется применять пресловутый подход "Copy/Paste", т. е. максимально использовать готовые решения в виде библиотек компонентов и функций, иногда без объяснения, откуда взялась та или иная функция. Надеемся, что вдумчивые читатели разберутся с этим сами.

Какие библиотеки компонентов мы будем использовать

Быстрая разработка приложений в Delphi основывается на повторном использовании кода других разработчиков и визуальном программировании с использованием

библиотек компонентов *VCL* (Visual Component Library). В Delphi имеется огромная библиотека стандартных компонентов и многие программисты только их и используют. Но возможностей стандартных компонентов бывает недостаточно и тысячи разработчиков занимаются собственными разработками, от отдельных компонентов до больших библиотек. Так как нам нужно спроектировать систему очень быстро, мы постараемся максимально использовать преимущества повторного кода. Из многотысячной номенклатуры библиотек компонентов мы будем отдавать предпочтение бесплатным библиотекам, поставляемым с исходными текстами.

Пожалуй, самой известной библиотекой типа *FWS* (Freeware With Source) являлась библиотека RxLib. Она была написана еще для Delphi 1 и за многие годы стала "классикой жанра". После ряда злоключений[1] RxLib вошла в состав библиотеки JVCL. Компоненты и функции из библиотеки JEDI[2] Visual Comonent Library (JVCL) и JEDI Code Library (JCL) мы и будем чаще всего использовать. Некоторые программисты называют эти библиотеки "свалкой" и доля истины в таком утверждении есть, т. к. библиотеки JEDI формируются за счет добровольных пожертвований авторов, а координаторы проекта не успевают все утрясти. Тем не менее это очень полезные сборники, особенно с учетом того, что в состав проекта JEDI вошла и библиотека RxLib.

Мы будем использовать и некоторые иные компоненты. Указания об этом будут приводиться в соответствующих главах.

Краткое знакомство с COM-технологиями

Модель компонентных[3] объектов *COM* (Component Object Model) — одна из базовых технологий, используемых в Windows. COM является основой таких технологий, как ActiveX, OLE, DirectX. Мы не будем совсем разбирать то, с чего начинается большинство описаний COM-технологий: *"В основе всех COM-интерфейсов лежит интерфейс с именем IUnknown. Он содержит указатели на три функции: AddRef, Release и QueryInterface. Любой COM-интерфейс должен содержать методы интерфейса IUnknown..."*. Действительно, *интерфейс* является основой COM и без полного понимания его сущности невозможно *профессиональное* программирование. А вот использовать COM прикладному программисту в практических целях можно почти ничего не зная о том, как интерфейсы устроены внутри. По крайней мере, первоначально.

Мы с вами уже много раз использовали COM-технологии в Visual LISP, вызывая `vlax`-функции. Например, функция `vlax-get-acad-object` вернет что-то наподобие `#<VLA-OBJECT IAcadApplication 00a88728>`. Это всего лишь визуальное отображение, сформированное системой AutoCAD, но в нем имеется обозначение `IAcadApplication`, а это не что иное, как наименование интерфейса с объектом приложения AutoCAD. Буква "I" в начале названия в соответствии с соглашением свидетельствует о том, что это именно интерфейс. Интерфейсы являются "стыковочными узлами" между

---

[1] Поучительную историю RxLib можно прочитать на сайте **www.delphiplus.org**.

[2] **jvcl.sourceforge.net**.

[3] Не следует путать компоненты в COM с компонентами, используемыми в Delphi. Компоненты в Delphi — это классы, из которых статически скомпоновано приложение. Компоненты COM — во-первых, это не классы, а во-вторых, они находятся вне файла приложения и могут изменяться содержательно при неизменности интерфейса.

приложениями и объектами, написанными разными разработчиками на разных языках, так же, как стыковочные узлы позволяют присоединяться к международной космической станции кораблям, изготовленным в разных странах. Сами корабли могут менять свою конструкцию, но стыковочный узел должен оставаться постоянным. Система AutoCAD тоже может как угодно изменять свое внутреннее устройство и реализацию используемых в нем классов, но пока интерфейс `IAcadApplication` остается неизменным, наше приложение будет с ним работать правильно.

В соответствии с технологией COM *приложение* может содержать один или несколько *объектов*. Приложение AutoCAD содержит множество объектов, наши приложения обычно будут содержать один объект. Каждый *объект* имеет один или несколько *интерфейсов*. Интерфейсы описывают *методы объекта*, к которым могут иметь доступ другие приложения[1]. Объекты находятся в *серверах COM* — EXE-файлах или DLL-библиотеках[2]. COM-сервер, реализованный в DLL, выполняется в процессе вызывающей программы (In-Proc сервер), а COM-сервер, реализованный в EXE-файла, выполняется в виде самостоятельного процесса (Out-Of-Proc сервер) или даже на другом компьютере (Distributed COM).

Приложение, обращающееся к COM-серверу, является *клиентом* COM. Клиент управляет сервером, вызывая его методы через указатель на интересующий интерфейс.

Что такое автоматизация в COM

В своих разработках мы будем использовать одно из подмножеств COM — *автоматизацию* (Automation).

Замечание

Microsoft до предела запутала терминологию COM-технологий. Термин *OLE*, ранее расшифровывавшийся как *Object Linking and Embedding*, теперь трактуется просто как клич *О-Ле!*. То, что ранее называлось *OLE-автоматизацией*, теперь входит в *ActiveX*. Но есть еще и *визуальные элементы управления ActiveX*, применяемые в Visual Basic, и *ActiveX Automation* — то, что мы используем в AutoCAD. Есть и *ActiveX-документы*, и *OLE-container*, и *OLE-object*. Есть *стандарты OLE1*, *OLE2* и *COM+*. Есть *модели COM* STA и MTA, *технологии-сателлиты* MTS и MSMQ. Будьте осторожны при чтении литературы разных лет издания.

Автоматизация — это способность приложения на программном уровне управлять объектами другого приложения или предоставлять свои функциональные возможности другим приложениям. Объекты автоматизации, а не обычные COM-объекты, необходимо создавать в приложениях, доступ к которым должен быть возможен и из приложений, создаваемых на интерпретирующих языках. А это именно наш случай, т. к. приложения Visual LISP, несмотря на то, что они компилируются в FAS- или

---

[1] Приложения VBA, VB и Delphi для AutoCAD имеют доступ к нему только через интерфейсы объектной модели, приложения Visual LISP имеют доступ не только через объектную модель, но и через команды и ent-функции. Приложения ObjectARX работают с AutoCAD не через объектную модель, но на Visual C++ можно разрабатывать приложения, работающие с AutoCAD через COM-интерфейсы.

[2] Не все DLL содержат COM-объекты.

VLX-формат, работают только внутри AutoCAD и не имеют доступа к специальной таблице, описывающей ссылки на методы объекта, реализующего интерфейс. За возможность использования автоматизации приходится платить.

- *Во-первых*, это снижение производительности при доступе к серверу автоматизации через переменные типа `VARIANT` (по сравнению с двумя другими возможными способами — посредством обычного интерфейса и интерфейса диспетчерирования).
- *Во-вторых*, это необходимость использования типов данных из числа поддерживаемых автоматизацией (табл. 15.1).

Таблица 15.1. *Типы данных, совместимых с автоматизацией.*

Тип данных Automation	Тип данных Delphi
`Boolean`	`WordBool`
`Unsigned Char`	`Byte`
`Double`	`Double`
`Float`	`Single`
`Int`	`SYSINT`
`Long`	`Integer`
`Short`	`SmallInt`
`BSTR`	`WideString`
`Currency`	`Currency`
`Date`	`TDateTime`
`SAFEARRAY`	`PSafeArray`
`Decimal`	`TDecimal`
`Interface IDispatch*`	`IDispatch`
`Interface IUnknown*`	`IUnknown`
`VARIANT`	`OleVariant`

- *В-третьих*, все методы должны быть процедурами или функциями, возвращающими значение типа `HRESULT`.
- *В-четвертых*, все методы должны иметь соглашение о вызовах `safecall`.

Как создать внутренний сервер автоматизации

Почти все наши объекты автоматизации будут находиться во внутренних серверах, т. е. в виде DLL. А что именно должны уметь делать эти объекты? Целесообразно использовать их в нескольких случаях.

- *Во-первых*, для создания приложений с пользовательским интерфейсом, который трудно или невозможно реализовать средствами DCL. Таких примеров вы увидите очень много.

- *Во-вторых*, для использования различных функций, которые нельзя написать на Visual LISP — вызов функций Windows API и т. п.
- *В-третьих*, для вызова функций, которые можно реализовать в Visual LISP, но с учетом масштаба нашей системы и наличия в ней приложений, написанных на разных языках, удобнее реализовывать в едином объекте, доступном из разных приложений. Примером может быть получение адресов размещения компонентов системы путем чтения реестра.

Создание сервера автоматизации в среде Delphi производится по следующей типовой схеме:

1. Создают DLL-библиотеку, которая будет являться субъектом автоматизации. Для этого в среде Delphi выбирают в меню **File | New | Other** (Файл | Новый | Другие), в появившемся диалоговом окне на вкладке **ActiveX** выбирают **ActiveX Library** (Библиотека ActiveX). Delphi создает проект, который сразу же, не меняя в нем ничего, сохраняют под именем будущего сервера, например MySvr.
2. Добавляют в созданный проект объект автоматизации. Для этого в среде Delphi выбирают в меню **File | New | Other** (Файл | Новый | Другие), в появившемся диалоговом окне на вкладке **ActiveX** выбирают **Automation Object** (Объект автоматизации). В появившемся диалоговом окне **Automation Object Wizard** (Мастер объекта автоматизации) в поле **CoClassName** (Имя класса) вводят имя объекта автоматизации, например, MyObject. Delphi открывает редактор библиотеки типов, в котором будет создана заготовка объекта автоматизации с именем `IMyObject`, производным от класса `TMyObject`.
3. В редакторе библиотеки типов добавляют в объект автоматизации свойства и методы. Библиотеку типов сохраняют в файл — будет предложено имя MySvr.tlb, при работе над проектом редактор библиотеки типов вызывают через меню **Project | View Type Library** (Проект | Просмотр библиотеки типов). После добавления или изменения свойств и методов щелкают кнопку **Refresh Implementation** (Обновить реализацию) и Delphi создает два файла — MySvr_TLB.pas (в этом файле никогда ничего не надо менять) и заготовку модуля реализации, который сохраняют, например, под именем u_MyObject.pas. В модуле реализации сгенерированы заготовки всех требуемых процедур и функций.
4. Редактируют модуль реализации, добавляя код реализации задуманных методов.
5. Компилируют проект. При отсутствии ошибок Delphi создает файл MySvr.dll. Для его использования нужно зарегистрировать COM-сервер. На этапе разработки это делают щелчком по кнопке **Register** (Регистрация) редактора библиотеки типов или с помощью команды OS `regsvr32 MySvr.dll`.

Полученный файл MySvr.dll будет поставляться пользователю, файл MySvr.tlb может поставляться для других разработчиков (шустрые обойдутся и без него), файл MySvr_TLB.pas может применяться при разработке других Delphi-приложений, в которых потребуется использование созданного сервера автоматизации.

Обычно мы, до выполнения этих основных этапов, разрабатываем простое приложение, в котором добиваемся правильной работы задуманных функций, а после создания сервера (при редактировании модуля реализации) используем в нем уже отлаженные формы и модули. После разработки сервера мы делаем специальное тестовое приложение, в котором проверяем работу сервера автоматизации при вызове из

другой Delphi-программы, и только в последнюю очередь, когда уже точно знаем, что все работает верно, разрабатываем LISP-функции.

Разработка первого COM-сервера

Попробуем разработать такие же функции, как и рассмотренные в *главе 14*, но с использованием COM-технологии. Напомним, что в *главе 14* были разработаны функции, которые из LISP-программ можно вызывать так:

- `(ruarx-set-ini <file_name> <section> <var_name> <value>)` — запись строки `<value>` в переменную `<var_name>` секции `<section>` файла `<file_name>`;
- `(ruarx-get-ini <file_name> <section> <var_name> <value>)` — чтение переменной `<var_name>` секции `<section>` файла `<file_name>` со значением по умолчанию `<value>`;
- `(ruarx-run-app <command_line> [<wait>])` — выполнение `<command_line>` (запуск программы, возможно с параметрами) и ожидание завершения с блокировкой AutoCAD, если `<wait>` равен `T`;
- `(ruarx-win-msg <header> <message> <style>)` — вывод стандартного диалогового окна.

Напомним так же, что полученные ARX-функции можно использовать только в LISP и только при загрузке ARX-файла, откомпилированного в соответствующей версии Visual C++.

Имена и синтаксис функций в COM-сервере, конечно, будут другие, но результат их работы в Visual LISP должен быть таким же. Наш COM-сервер будет содержать три объекта автоматизации: работа с INI-файлами, запуск приложений и вывод диалоговых окон.

Итак, запускаем Borland Delphi и приступаем к реализации задуманного.

Шаг 1. В среде Borland Delphi мы выбираем пункт меню **File | New** (Файл | Новый).

Шаг 2. В появившемся диалоговом окне **New Items** на вкладке **ActiveX** выбираем **ActiveX Library** (Библиотека ActiveX) (рис. 15.1).

Шаг 3. Сохраняем созданный проект под именем ruAxSvr.dpr.

Шаг 4. Выбираем пункт меню **File | New** (Файл | Новый). На вкладке **ActiveX** выбираем **Automation Object** (Объект автоматизации) (рис. 15.2).

Шаг 5. В открывшемся диалоговом окне **Automation Object** (Объект автоматизации) (рис. 15.3) в поле **CoClassName** (Имя класса) вводим `RunApp` и щелкаем по кнопке **OK**. Открывается окно редактора библиотеки типов с корневым элементом `ruAxSvr`, имеющим в наследниках `interface IRunApp CoClass RunApp`.

Повторяем еще два раза шаг 4, вводя имена объектов `WinMsg` и `IniFile`. После создания третьего объекта окно редактора библиотеки типов будет выглядеть, как показано на рис. 15.4.

Шаг 6. Переходим к наполнению объектов автоматизации методами. Свойств для таких объектов не требуется, можно только создать несколько методов. Для создания метода необходимо выделить интерфейс объекта (на рис. 15.4 выделен `IRunApp`)

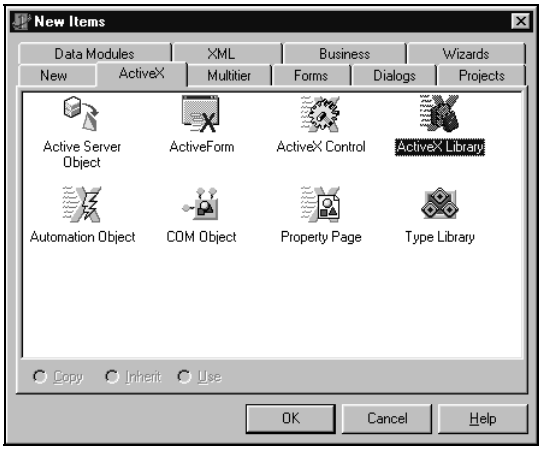


Рис. 15.1. Создание ActiveX Library

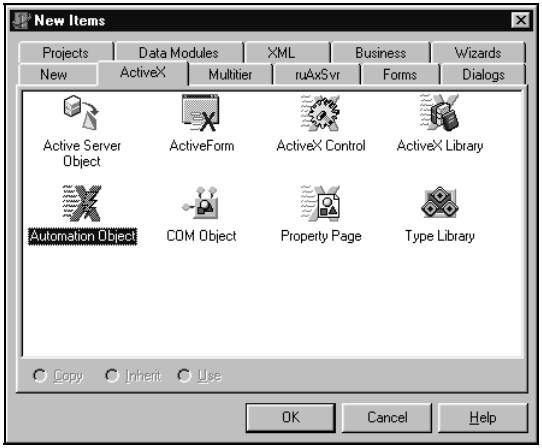


Рис. 15.2. Создание объекта автоматизации

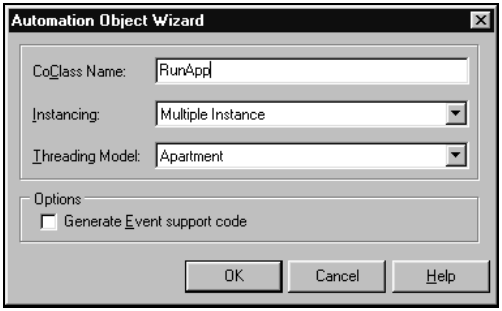


Рис. 15.3. Ввод имени объекта

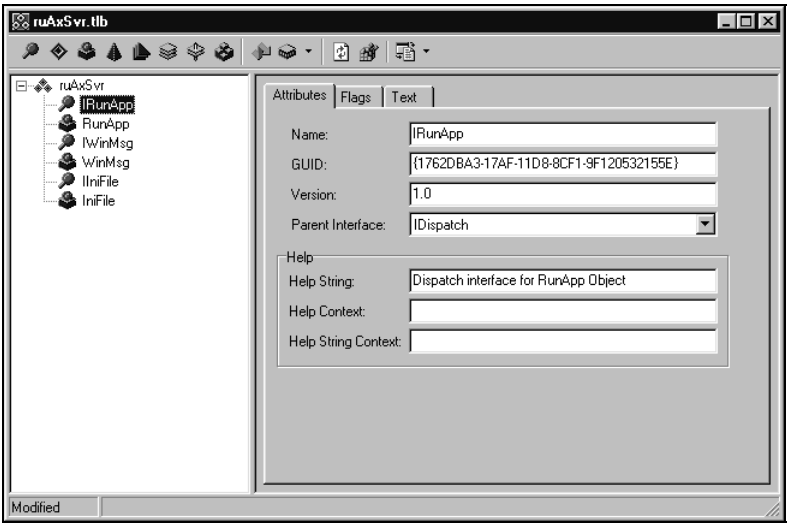


Рис. 15.4. Редактор библиотеки типов после создания объектов автоматизации

и добавить метод из панели инструментов или из контекстного меню выбором **New | Method** (Новый | Метод). После добавления имя метода по умолчанию заменяется на требуемое.

В нашем приложении мы создадим такие методы:

- для `IRunApp`:
  - метод `Run` — запуск приложения;
- для `IWinMsg`:
  - метод `Show` — вывод на экран диалогового окна;
- для `IIniFile`:
  - метод `ReadString` — чтение строковой переменной;
  - метод `WriteString` — запись строковой переменной.

Теперь редактор библиотеки типов должен выглядеть так, как показано на рис. 15.5.

А теперь найдите два различия между рис. 15.4 и 15.5, не считая добавленных методов. На рис. 15.4 интерфейс для INI-файлов называется `IIniFile`, а на рис. 15.5 — уже `IruIniFlile`. Объясняется это тем, что мы (в данном случае намеренно) допустили ошибку при назначении имени. Имя `IniFile` приведет к тому, что в проекте появится класс `TIniFile`, но именно такое имя у стандартного класса для работы с INI-файлами. Два одноименных класса не могут сосуществовать в проекте, поэтому мы переименовали прямо в редакторе библиотеки типов `IniFile` на `ruIniFile`. Если бы этого мы не сделали сейчас, ошибка выявилась бы позже.

Шаг 7. Вводим параметры методов (рис. 15.6).

Для этого выделяем метод, переходим на вкладку **Parameters** (Параметры), щелчком по кнопке **Add** (Добавить) добавляем параметр, устанавливаем **Type** (Тип) и **Modifier** (Модификатор). Тип параметра можно выбрать из списка или ввести вручную из

перечня допустимых (см. табл. 15.1). У нас практически все параметры будут строкового типа. В редакторе библиотеки типов нужно выбирать тип `BSTR` для строк и тип `VARIANT_BOOL` для логических параметров. Менять тип **Return Type** (Тип возвращаемого значения) с именем `HRESULT` на иной тип не следует. Для входных параметров должен быть установлен `Modifier [in]`. Только один, причем последний параметр может иметь `Modifier[out, retval]`.

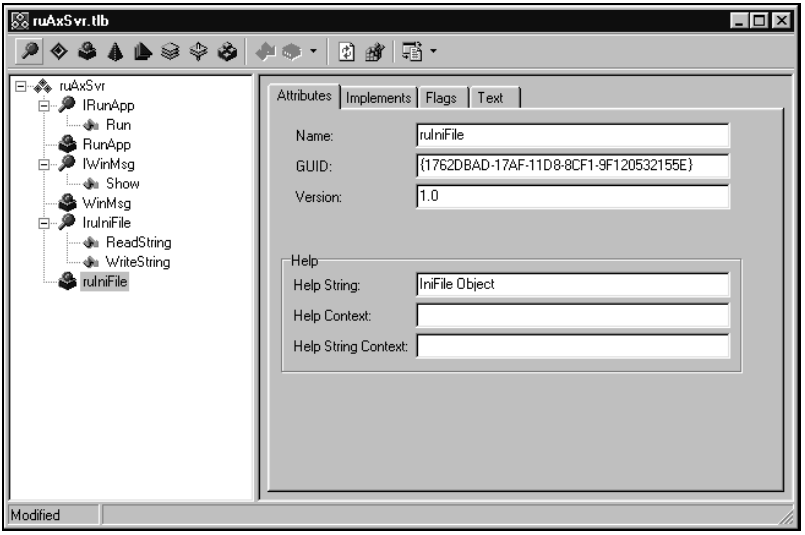


Рис. 15.5. Редактор библиотеки типов после добавления методов

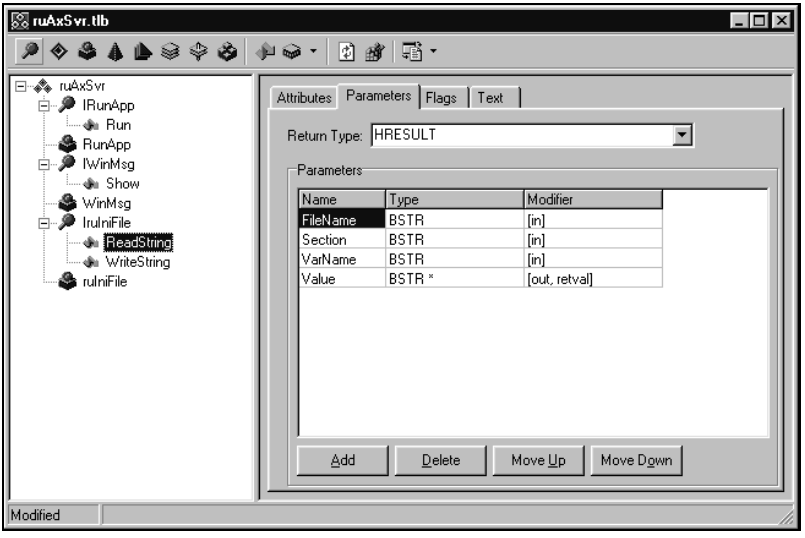


Рис. 15.6. Редактирование параметров методов

Для наглядности сведем все методы и параметры в табл. 15.2.

Таблица 15.2. *Параметры методов COM-сервера*

Объект	Метод	Параметр	Тип	Модификатор
RunApp	Run	CommandLine	BSTR	[in]
RunApp	Run	Wait	VARIANT_BOOL	[in]
WinMsg	Show	Caption	BSTR	[in]
WinMsg	Show	Text	BSTR	[in]
WinMsg	Show	Style	Long	[in]
WinMsg	Show	Result	Long*	[out, retval]
ruIniFile	ReadString	FileName	BSTR	[in]
ruIniFile	ReadString	Section	BSTR	[in]
ruIniFile	ReadString	VarName	BSTR	[in]
ruIniFile	ReadString	Default	BSTR	[in]
ruIniFile	ReadString	Result	BSTR*	[out, retval]
ruIniFile	WriteString	FileName	BSTR	[in]
ruIniFile	WriteString	Section	BSTR	[in]
ruIniFile	WriteString	VarName	BSTR	[in]
ruIniFile	WriteString	Value	BSTR	[in]

Шаг 8. После завершения редактирования библиотеки типов нужно щелкнуть по кнопке **Refresh Implementation** (Обновить реализацию) и Delphi автоматически сгенерирует файл ruAxSvr_TLB.pas и три модуля реализации, которые надо сохранить под именами u_RunApp.pas, u_WinMsg.pas и u_IniFile.pas.

Шаг 9. В файле ruAxSvr_TLB.pas ничего менять не нужно. Он предназначен для использования в других программах, обращающихся к нашему объекту автоматизации. Изучить его следует хотя бы для того, чтобы понять, какой объем работы выполнен автоматически. Полезно разобраться, как преобразуются типы данных из библиотеки типов в типы языка Delphi (листинг 15.1, в котором для экономии места удалены объемные комментарии). Обратите внимание на то, сколько "умного кода" сгенерировано автоматически! Когда-то его приходилось писать вручную, а мы, с нулевой теоретической подготовкой, получили его нажатием нескольких кнопок. Конечно лучше, если программист хотя бы знает, что именно получено, но ответы на множество вопросов надо искать в специализированных изданиях. Обратите внимание на то, как Delphi преобразовала объявления методов в библиотеке типов в соответствующие объявления языка Delphi.

Листинг 15.1. Файл ruAxSvr_TLB.pas

```
unit ruAxSvr_TLB;

{$TYPEDADDRESS OFF}
interface
```

```
uses Windows, ActiveX, Classes, Graphics, OleServer, OleCtrls, StdVCL;

const
ruAxSvrMajorVersion = 1;
  ruAxSvrMinorVersion = 0;

  LIBID_ruAxSvr: TGUID = '{1762DBA2-17AF-11D8-8CF1-9F120532155E}';

  IID_IRunApp: TGUID = '{1762DBA3-17AF-11D8-8CF1-9F120532155E}';
  CLASS_RunApp: TGUID = '{1762DBA5-17AF-11D8-8CF1-9F120532155E}';
  IID_IWinMsg: TGUID = '{1762DBA7-17AF-11D8-8CF1-9F120532155E}';
  CLASS_WinMsg: TGUID = '{1762DBA9-17AF-11D8-8CF1-9F120532155E}';
  IID_IruIniFile: TGUID = '{1762DBAB-17AF-11D8-8CF1-9F120532155E}';
  CLASS_ruIniFile: TGUID = '{1762DBAD-17AF-11D8-8CF1-9F120532155E}';
type

IRunApp = interface;
  IRunAppDisp = dispinterface;
  IWinMsg = interface;
  IWinMsgDisp = dispinterface;
  IruIniFile = interface;
  IruIniFileDisp = dispinterface;

  RunApp = IRunApp;
  WinMsg = IWinMsg;
  ruIniFile = IruIniFile;


  IRunApp = interface(IDispatch)
    ['{1762DBA3-17AF-11D8-8CF1-9F120532155E}']
    procedure Run(const CommandLine: WideString; Wait: WordBool); safecall;
  end;

  IRunAppDisp = dispinterface
    ['{1762DBA3-17AF-11D8-8CF1-9F120532155E}']
    procedure Run(const CommandLine: WideString; Wait: WordBool); dispid 2;
  end;

  IWinMsg = interface(IDispatch)
    ['{1762DBA7-17AF-11D8-8CF1-9F120532155E}']
    function  Show(const Caption: WideString; const Text: WideString; Style:
Integer): Integer; safecall;
  end;

  IWinMsgDisp = dispinterface
    ['{1762DBA7-17AF-11D8-8CF1-9F120532155E}']
    function  Show(const Caption: WideString; const Text: WideString; Style:
Integer): Integer; dispid 1;
  end;

  IruIniFile = interface(IDispatch)
    ['{1762DBAB-17AF-11D8-8CF1-9F120532155E}']
    function  ReadString(const FileName: WideString; const Section: WideString;
                         const VarName: WideString; const Default: WideString):
                                                              WideString; safecall;
```

```
    procedure WriteString(const FileName: WideString; const Section: WideString;
                          const VarName: WideString; const Value: WideString);
safecall;
  end;

  IruIniFileDisp = dispinterface
    ['{1762DBAB-17AF-11D8-8CF1-9F120532155E}']
    function  ReadString(const FileName: WideString; const Section: WideString;
                         const VarName: WideString; const Default: WideString):
WideString; dispid 1;
    procedure WriteString(const FileName: WideString; const Section: WideString;
                          const VarName: WideString; const Value: WideString);
dispid 2;
  end;

  CoRunApp = class
    class function Create: IRunApp;
    class function CreateRemote(const MachineName: string): IRunApp;
  end;

  CoWinMsg = class
    class function Create: IWinMsg;
    class function CreateRemote(const MachineName: string): IWinMsg;
  end;

  CoruIniFile = class
    class function Create: IruIniFile;
    class function CreateRemote(const MachineName: string): IruIniFile;
  end;

implementation

uses ComObj;

class function CoRunApp.Create: IRunApp;
begin
  Result := CreateComObject(CLASS_RunApp) as IRunApp;
end;

class function CoRunApp.CreateRemote(const MachineName: string): IRunApp;
begin
  Result := CreateRemoteComObject(MachineName, CLASS_RunApp) as IRunApp;
end;

class function CoWinMsg.Create: IWinMsg;
begin
  Result := CreateComObject(CLASS_WinMsg) as IWinMsg;
end;

class function CoWinMsg.CreateRemote(const MachineName: string): IWinMsg;
begin
  Result := CreateRemoteComObject(MachineName, CLASS_WinMsg) as IWinMsg;
end;
```

```
class function CoruIniFile.Create: IruIniFile;
begin
  Result := CreateComObject(CLASS_ruIniFile) as IruIniFile;
end;

class function CoruIniFile.CreateRemote(const MachineName: string): IruIniFile;
begin
  Result := CreateRemoteComObject(MachineName, CLASS_ruIniFile) as IruIniFile;
end;

end.
```

Шаг 10. Нам осталось реализовать объявленные методы (листинги 15.2, 15.3, 15.4). У нас имеются три модуля реализации, соответствующие трем объектам автоматизации. Реализация методов очень проста и не требует объемных пояснений.

Листинг 15.2. Файл u_RunApp.pas

```
unit u_RunApp;

interface

uses
  ComObj, ActiveX, ruAxSvr_TLB, StdVcl;

type
  TRunApp = class(TAutoObject, IRunApp)
  protected
    procedure Run(const CommandLine: WideString; Wait: WordBool); safecall;
    { Protected declarations }
  end;

implementation

uses ComServ, JclMiscel;

procedure TRunApp.Run(const CommandLine: WideString; Wait: WordBool);
begin
if Wait then
  WinExec32AndWait(CommandLine,1)
else
 WinExec32(CommandLine,1);
end;

initialization
  TAutoObjectFactory.Create(ComServer, TRunApp, Class_RunApp,
    ciMultiInstance, tmApartment);
end.
```

Листинг 15.3. Файл u_WinMsg.pas

```
unit u_WinMsg;

interface
```

```
uses
  ComObj, ActiveX, ruAxSvr_TLB, StdVcl;

type
  TWinMsg = class(TAutoObject, IWinMsg)
  protected
    function Show(const Caption, Text: WideString; Style: Integer): Integer;
      safecall;
    { Protected declarations }
  end;

implementation

uses ComServ, Forms;

function TWinMsg.Show(const Caption, Text: WideString;
  Style: Integer): Integer;
  var
   Capt, Txt : string;
Begin
{Это промежуточное присваивание необходимо, иначе строки не отобразятся правильно}
 Capt:=Caption;
 Txt:=Text;
 Result := Application.MessageBox(PChar(Txt), PChar(Capt),Style);
end;

initialization
  TAutoObjectFactory.Create(ComServer, TWinMsg, Class_WinMsg,
    ciMultiInstance, tmApartment);
end.
```

Листинг 15.4. Файл u_IniFile.pas

```
unit u_IniFile;

interface

uses
  ComObj, ActiveX, ruAxSvr_TLB, StdVcl;

type
  TruIniFile = class(TAutoObject, IruIniFile)
  protected
    function ReadString(const FileName, Section, VarName,
      Default: WideString): WideString; safecall;
    procedure WriteString(const FileName, Section, VarName, Value: WideString);
      safecall;
    { Protected declarations }
  end;

implementation

uses ComServ, abIniFiles;
{
```

```
Вместо стандартного модуля IniFiles мы используем модуль abIniFiles
Автор: Антон Белов (www:abhere.by.ru)
Модуль совместим со стандартным IniFiles и имеет массу достоинств:
- физический размер файла не ограничен 64 Кбайт
- умеет записывать и читать интернациональный float правильно
- умеет записывать и читать положение формы, тексты TEdit и т. п.
- может создаваться из потока (TStream)
- имеется МНОГО разных хороших методов и свойств
- автономен. Не требуются нестандартные библиотеки
Позволяет преодолеть известную ошибку стандартного чтения INI в некоторых версиях
Windows, связанную с кэшированием данных, когда читаются устаревшие данные
}
function TruIniFile.ReadString(const FileName, Section, VarName,
  Default: WideString): WideString;
var
  Ini : TIniFile;
begin
  Ini:=TIniFile.Create(FileName);
  Result:=Ini.ReadString(Section,VarName, Default);
  Ini.Free;
end;

procedure TruIniFile.WriteString(const FileName, Section, VarName,
  Value: WideString);
var
  Ini : TIniFile;
begin
  Ini:=TIniFile.Create(FileName);
  Ini.WriteString(Section,VarName, Value);
  Ini.Free;
end;

initialization
  TAutoObjectFactory.Create(ComServer, TruIniFile, Class_ruIniFile,
    ciMultiInstance, tmApartment);
end.
```

Шаг 11. Компилируем проект, получаем файл ruAxSvr.dll.

Шаг 12. Регистрируем COM-сервер. Возможные варианты:

- Из командной строки операционной системы

  ```
  Regsvr32 ruAxSvr.dll
  ```

- Из меню **View | Type Library** (Вид | Библиотека типов) вызвать редактор библиотеки типов и щелкнуть по кнопке **Register Type Library** (Регистрация библиотеки типов).
- Из меню Delphi **Run | Register ActiveX Server** (Выполнить | Регистрация сервера ActiveX).

Теперь наш сервер готов к работе.

Изменения LISP-библиотеки

Теперь изменим библиотечные LISP-функции, которые мы написали в *главе 14*. Мы можем отказаться и от использования нашей библиотеки ru_MainLib.arx и от DOSLib (по крайней мере, для рассмотренных случаев). Конечно, функции COM-сервера работают медленнее, но для нас это не критично. Взамен мы получаем универсальность применения в любых программах. Новые и измененные функции приведены в листингах 15.5—15.13. Исключены прежние комментарии, выделены шрифтом изменения.

Листинг 15.5. Новая функция _ru-app-run-srv

```
(defun _ru-app-run-srv ()
;;; Создание COM-сервера для запуска приложений
  (vlax-get-or-create-object "ruAxSvr.RunApp")
)
```

Листинг 15.6. Новая функция _ru-ini-srv

```
(defun _ru-ini-srv ()
;;; Создание COM-сервера для работы с INI-файлами
  (vlax-get-or-create-object "ruAxSvr.ruIniFile")
);_ end of defun
```

Листинг 15.7. Новая функция _ru-msg-srv

```
(defun _ru-msg-srv ()
;;; Создание сервера для стандартных диалоговых окон
  (vlax-get-or-create-object "ruAxSvr.WinMsg")
);_ end of defun
```

Листинг 15.8. Измененная функция ru-app-run

```
(defun ru-app-run (command_line wait / srv)
(if (setq srv (_ru-app-run-srv))
    (vlax-invoke-method srv "Run" command_line
      (ru-conv-value-to-wordbool wait)
   );_ end of vlax-invoke-method
    (alert "Не удалось запустить ruAxSvr.RunApp")
 );_ end of if
);_ end of defun
```

Листинг 15.9. Новая функция ru-msg-srv-show

```
(defun ru-msg-srv-show (header text style / srv)
;;; Вывод диалогового окна
  (if (setq srv (_ru-msg-srv))
    (vlax-invoke-method srv "Show" header text style)
    (alert "Не удалось запустить ruAxSvr.WinMsg")
 );_ end of if
);_ end of defun
```

Листинг 15.10. Измененная функция `ru-msg-box-ex`

```
(defun ru-msg-box-ex (header text buttons_set_name icon_name
 default_button_number / win_btn_number win_def_btn_number
 win_icon_number result)
  (cond
    ((= buttons_set_name "OK_CANCEL")
     (setq win_btn_number vlax-vbokcancel)
    )
    ((= buttons_set_name "ABORT_RETRY_IGNORE")
     (setq win_btn_number vlax-vbabortretryignore)
    )
    ((= buttons_set_name "YES_NO_CANCEL")
     (setq win_btn_number vlax-vbyesnocancel)
    )
    ((= buttons_set_name "YES_NO")
     (setq win_btn_number vlax-vbyesno)
    )
    ((= buttons_set_name "RETRY_CANCEL")
     (setq win_btn_number vlax-vbretrycancel)
    )
    (T
     (setq win_btn_number vlax-vbokonly)
    )
 );_ end of cond
  (cond
    ((= icon_name "ICONSTOP")
     (setq win_icon_number 16)
   )
    ((= icon_name "ICONQUESTION")
     (setq win_icon_number vlax-vbquestion)
   )
    ((= icon_name "ICONEXCLAMATION")
     (setq win_icon_number vlax-vbexclamation)
   )
    ((= icon_name "ICONINFORMATION")
     (setq win_icon_number vlax-vbinformation)
   )
    (T
     (setq win_icon_number 0)
   )
 );_ end of cond
  (cond
    ((= default_button_number 2)
     (setq win_def_btn_number vlax-vbdefaultbutton2)
   )
    ((= default_button_number 3)
     (setq win_def_btn_number vlax-vbdefaultbutton3)
   )
    (T
     (setq win_def_btn_number vlax-vbdefaultbutton1)
   )
 );_ end of cond
```

```
  (setq result (ru-msg-srv-show;|ruarx-win-msg|;
  header text (logior win_btn_number win_icon_number win_def_btn_number))
 )
  (cond
    ((= result vlax-vbok) "OK")
    ((= result vlax-vbcancel) "CANCEL")
    ((= result vlax-vbabort) "ABORT")
    ((= result vlax-vbretry) "RETRY")
    ((= result vlax-vbignore) "IGNORE")
    ((= result vlax-vbyes) "YES")
    ((= result vlax-vbno) "NO")
    ((= result 8) "CLOSE")
    ((= result 9) "HELP")
    (T nil)
 );_ end of cond
);_ end of defun
```

Листинг 15.11. Измененная функция ru-msg-messagebox

```
(defun ru-msg-messagebox (header text buttons_set_name icon_name
default_button_number)
;|(if *ru_use_ru_arx* |;
(ru-msg-box-ex header text buttons_set_name icon_name
       default_button_number)
;|(ru-msg-dos-msgboxex header text buttons_set_name icon_name 1)
;;)|;
);_ end of defun
```

Листинг 15.12. Измененная функция ru-ini-read

```
(defun ru-ini-read (ini_name section var default / srv)
(if (setq srv (_ru-ini-srv))
    (vlax-invoke-method srv "ReadString" ini_name section var default)
    (alert "Не удалось запустить ruAxSvr.ruIniFile")
 );_ end of if
);_ end of defun
```

Листинг 15.13. Измененная функция ru-ini-write

```
(defun ru-ini-write (ini_name section var string / srv)
(if (setq srv (_ru-ini-srv))
    (vlax-invoke-method srv "WriteString" ini_name section var string)
    (alert "Не удалось запустить ruAxSvr.ruIniFile")
 );_ end of if
);_ end of defun
```

Благодаря заранее продуманной системе сокрытия низкоуровневых функций нам удалось, немного откорректировав несколько библиотечных функций и не затрагивая остальных, перейти на принципиально новую технологию.

Другие COM-серверы в ruCAD

Мы подробно рассмотрели создание одного внутреннего COM-сервера с тремя объектами автоматизации. В последующих главах будут рассмотрены еще несколько самых интересных реальных примеров:

- *глава 16* — работа с XML-меню и таблицами;
- *глава 17* — диалоги выбора файлов и папок;
- *глава 19* — работа с базами данных;
- *глава 21* — диалог-мастер вычерчивания формата.

Остальные примеры мы разбирать не будем, приведем только краткие описания, иллюстрации, фрагменты модулей реализации и LISP-функции, использующие эти COM-объекты.

Заставка с сообщением для длительных операций

Выводится для предупреждения пользователя о выполнении длительной операции (рис. 15.7). Возможно изменение текста во время операции.

Библиотека — ruSplashSvr.dll, объект `ruSplashSvr.Splash`, объявление методов см. в листинге 15.14, функции для LISP — в листингах 15.15—15.18.

Листинг 15.14. Объявление объекта `ruSplashSvr.Splash`

```
type
  TSplash = class(TAutoObject, ISplash)
  protected
    function Show(msg: OleVariant): OleVariant; safecall;
    function SetText(msg: OleVariant): OleVariant; safecall;
    function Hide: OleVariant; safecall;
  end;
```

Листинг 15.15. Функция `_ru-dlg-splash-srv`

```
(defun _ru-dlg-splash-srv ()
;;; Создание объекта заставки
 (vlax-get-or-create-object "ruSplashSvr.Splash")
)
```

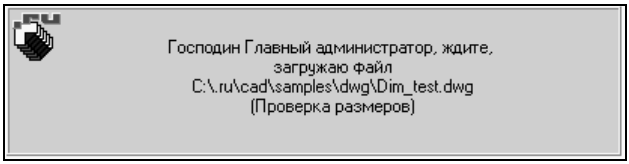


Рис. 15.7. Вывод заставки

Листинг 15.16. Функция ru-splash-show

```
(defun ru-splash-show (msg / srv)
;;; Вывод заставки с сообщением msg
  (if (setq srv (_ru-dlg-splash-srv))
   (vlax-invoke-method srv "Show" msg)
   (princ (strcat "\nОшибка Splash. Сообщение: " msg))
 );_ end of if
)
```

Листинг 15.17. Функция ru-splash-set-text

```
(defun ru-splash-set-text (msg / srv)
;;; Изменение сообщения в заставке
  (if (setq srv (_ru-dlg-splash-srv))
   (vlax-invoke-method srv "SetText" msg)
   (princ (strcat "\nОшибка Splash. Сообщение: " msg))
 );_ end of if
)
```

Листинг 15.18. Функция ru-splash-hide

```
(defun ru-splash-hide (/ srv)
;;; Удаление заставки с экрана
  (if (setq srv (_ru-dlg-splash-srv))
   (vlax-invoke-method srv "Hide")
 );_ end of if
)
```

Диалоговое окно выбора из одинарного списка

Это диалоговое окно часто применяется для выбора строки из списка (см. *главу 19*, рис. 19.8).

Библиотека — ruSingleListSrv.dll, объект `ruSingleListSrv.SingleListDlg`, объявление методов см. в листинге 15.19, функцию для LISP — в листинге 15.20.

Листинг 15.19. Объявление объекта ruSingleListSrv.SingleListDlg

```
type
  TSingleListDlg = class(TAutoObject, ISingleListDlg)
  protected
    function Execute: OleVariant; safecall;
    function Get_ResultString: WideString; safecall;
    procedure Set_DlgCaption(const Value: WideString); safecall;
    procedure Set_ListCaption(const Value: WideString); safecall;
    procedure Set_ResultString(const Value: WideString); safecall;
    procedure AddStringToList(const Value: WideString); safecall;
    procedure Create; safecall;
    procedure Free; safecall;
```

```
    { Protected declarations }
  end;
```

Листинг 15.20. Функция `ru-dlg-single-list`

```
(defun ru-dlg-single-list (dlg_caption lst_caption lst_values /
                                res_srv result srv)
;;; Вывод списка строк для выбора
;|
Аргументы:
dlg_caption - заголовок окна
lst_caption - заголовок списка
lst_values - список строк
Возвращает выбранную строку или NIL
Пример:
(ru-dlg-single-list "Проверка"  "Заголовок списка"
  (list "Переменная 1" "Переменная 2" "Переменная 3")
)
|;
(if (setq srv (vlax-get-or-create-object "ruSingleListSrv.SingleListDlg"))
 (progn
   (vlax-invoke-method srv "Create")
   (vlax-put-property srv 'DlgCaption dlg_caption)
   (vlax-put-property srv 'ListCaption lst_caption)
   (foreach s lst_values
     (vlax-invoke-method srv "AddStringToList" s)
  );_ end of foreach
   (setq res_srv (vlax-invoke-method srv "Execute"))
   (if (= (vlax-variant-value res_srv) :vlax-true)
      (setq result (vlax-get-property srv 'ResultString))
  );_ end of if
  (vlax-invoke-method srv "Free")
);_ end of progn
 (ru-msg-alert "Не удалось запустить SingleListDlg")
);_ end of if
 result
)
```

Список из двух колонок с возможностью редактирования

В этом диалоговом окне выводится двухколоночный список (*глава 19*, рис. 19.5) с параметрами и их значениями. Значения, факультативно, можно редактировать.

Библиотека — ruDoubleListSrv.dll, объект `ruDoubleListSrv.DoubleListDlg`, объявление методов см. в листинге 15.21, функцию для LISP — в листинге 15.22.

Листинг 15.21. Объявление объекта `ruDoubleListSrv.DoubleListDlg`

```
type
  TDoubleListDlg = class(TAutoObject, IDoubleListDlg)
```

```
  protected
    procedure AddLeftString(const Value: WideString); safecall;
    procedure AddRightString(const Value: WideString); safecall;
    procedure Create; safecall;
    function Execute: OleVariant; safecall;
    procedure Free; safecall;
    procedure Set_Header(const Value: WideString); safecall;
    procedure Set_LeftCaption(const Value: WideString); safecall;
    procedure Set_ReadOnly(Value: WordBool); safecall;
    procedure Set_RightCaption(const Value: WideString); safecall;
    function Get_LeftResultString: WideString; safecall;
    function Get_RightResultString: WideString; safecall;
    { Protected declarations }
  end;
```

Листинг 15.22. Функция `ru-dlg-double-list`

```
(defun ru-dlg-double-list (dlg_caption left_caption right_caption
              lst_vars lst_values read_only / res_srv result srv)
;;; Двойной список с редактированием
;|
Аргументы:
dlg_caption - заголовок окна
left_caption - заголовок левого списка
right_caption - заголовок правого списка
lst_vars - левый список переменных
lst_values - правый список значений
read_only - режим только для чтения?
Пример:
(ru-dlg-double-list "Проверка" "Переменная" "Значение"
  (list "Переменная 1" "Переменная 2" "Переменная 3")
  (list "Значение 1" "Значение 2" "Значение 3")
  nil
)
|;
 (if (setq srv (vlax-get-or-create-object
         "ruDoubleListSrv.DoubleListDlg"))
 (progn
   (vlax-invoke-method srv "Create")
   (vlax-put-property srv 'Header dlg_caption)
   (vlax-put-property srv 'LeftCaption left_caption)
   (vlax-put-property srv 'RightCaption right_caption)
   (vlax-put-property srv 'ReadOnly (ru-conv-value-to-wordbool
      read_only))
   (foreach s lst_vars
    (vlax-invoke-method srv "AddLeftString" s)
  );_ end of foreach
   (foreach s lst_values
    (vlax-invoke-method srv "AddRightString" s)
  )
```

```
   (setq res_srv  (vlax-invoke-method srv "Execute"))
   (if (= (vlax-variant-value res_srv) :vlax-true)
     (setq result (list (vlax-get-property srv 'LeftResultString)
     (vlax-get-property srv 'RightResultString)))
  );_ end of if
   (vlax-invoke-method srv "Free")
);_ end of progn
(ru-msg-alert "Не удалось запустить SingleListDlg")
);_ end of if
result
)
```

Двухоконный список

Двухоконный список (рис. 15.8) позволяет отобрать строки из двух списков и сформировать один (с возможностью регулирования порядка следования в списке перетаскиванием).

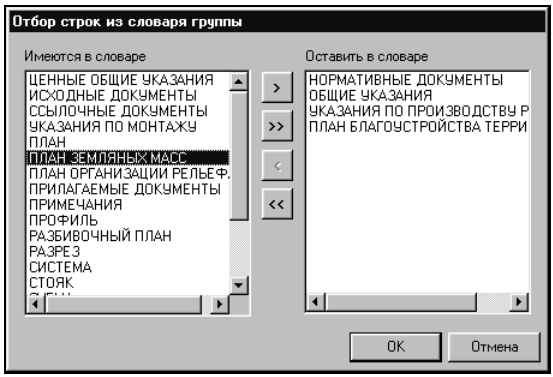


Рис. 15.8. Двухоконный список

Библиотека — ruDualListSrv.dll, объект `ruDualListSrv.ruCadDualLst`, объявление методов см. в листинге 15.23, функцию для LISP — в листинге 15.24.

Листинг 15.23. Объявление объекта `ruDualListSrv.ruCadDualLst`

```
type
  TruCadDualLst = class(TAutoObject, IruCadDualLst)
  protected
    function Get_LeftList: IStrings; safecall;
    function Get_RightList: IStrings; safecall;
    procedure Set_DlgCaption(const Value: WideString); safecall;
    procedure Set_LeftCaption(const Value: WideString); safecall;
    procedure Set_LeftList(const Value: IStrings); safecall;
    procedure Set_RightCaption(const Value: WideString); safecall;
    procedure Set_RightList(const Value: IStrings); safecall;
    procedure Execute; safecall;
```

```
    procedure Create; safecall;
    procedure Free; safecall;
    procedure Select(const DlgCaption, LeftCaption: WideString;
      var LeftList: IStrings; const RightCaption: WideString;
      var RightList: IStrings); safecall;
    procedure AddLeftString(const Value: WideString); safecall;
    procedure AddRightString(const Value: WideString); safecall;
    procedure GetLeftStringList(out value: OleVariant); safecall;
    procedure GetRightStringList(out value: OleVariant); safecall;
    function Get_LeftListCount: Integer; safecall;
    function Get_RightListCount: Integer; safecall;
    procedure GetLeftString(Index: Integer; out Value: OleVariant); safecall;
    procedure GetRightString(Index: Integer; out Value: OleVariant); safecall;
  end;
```

Листинг 15.24. Функция `ru-dlg-dual-list`

```
(defun ru-dlg-dual-list (dlg_caption left_caption right_caption left_list
                         right_list / i left_count right_count s srv)
;;; Выбор из двухоконного списка
;|
Аргументы:
dlg_caption - заголовок диалогового окна
left_caption - заголовок левого списка
right_caption - заголовок правого списка
left_list - левый список
right_list - правый список
Возвращает список из левого и правого списка
|;
 (if (setq srv (vlax-get-or-create-object "ruDualListSrv.ruCadDualLst"))
  (progn
    (vlax-invoke-method srv "Create")
    (vlax-put-property srv 'DlgCaption dlg_caption)
    (vlax-put-property srv 'LeftCaption left_caption)
    (vlax-put-property srv 'RightCaption right_caption)
    (foreach s left_list (vlax-invoke-method srv "AddLeftString" s))
    (foreach s right_list (vlax-invoke-method srv "AddRightString" s))
    (vlax-invoke-method srv "Execute")
    (setq left_count (vlax-get-property srv 'LeftListCount)
    left_list '() i 0)
    (repeat left_count
     (setq s "")
     (vlax-invoke-method srv "GetLeftString" i 's)
     (setq left_list (cons s left_list) i (1+ i))
   )
    (setq left_list (reverse left_list) right_list '() i 0
       right_count (vlax-get-property srv 'RightListCount))
     (repeat right_count
       (setq s "")
       (vlax-invoke-method srv "GetRightString" i 's)
```

```
      (setq right_list (cons s right_list) i (1+ i))
    );_ end of repeat
     (setq right_list (reverse right_list))
     (vlax-invoke-method srv "Free");_ end of vlax-invoke-method
   );_ end of progn
    (alert "Не удалось запустить ruDualListSrv.ruCadDualLst")
 );_ end of if
  (list left_list right_list)
)
```

У читателей, внимательно изучивших код, может возникнуть вопрос о том, почему же мы не используем один метод `Select`, а сначала инициализируем объект, затем построчно передаем ему строки левого и правого списков, а потом так же построчно их читаем. Именно такую схему мы применяем всегда, когда надо передать список строк.

Объясняется это тем, что объекту автоматизации список строк передать из LISP непросто из-за несоответствия типов. Мы можем передать список только с типом `OleVariant` в Delphi. В LISP мы можем только создать безопасный массив из строк и преобразовать его в тип `VARIANT`. Но такой тип данных COM-сервер не примет! В Delphi очень часто используются списки строк и для них введен специальный интерфейс `IStrings`. Имеется и немало вариантов преобразования типов, но из LISP они недоступны. Конечно, после ряда экспериментов проблему можно решить, но мы делаем реальную систему в условиях ограниченных сроков и, не задерживаясь на частностях, стремимся выполнить задуманное в срок. И это не такое уж плохое решение. Преобразования "из принципа" списков строк в безопасный массив и в данные типа `VARIANT` в LISP с последующим обратным преобразованием в DLL займут, вероятно, не меньше времени.

Возможен вариант передачи не очень длинного списка строк в виде одной длинной строки с разделителями, наподобие `ConnectionString`, рассматривать которую мы будем в *главе 19*. Очень длинные списки возможно передавать через файл. Метод `Select` удобно применять в Delphi-приложениях.

Кстати, *получить* результат сразу в виде списка можно и в LISP:

```
(vlax-invoke-method srv "GetLeftStringList" 'left_safe_arr)
(vlax-invoke-method srv "GetRightStringList"'right_safe_arr)
(vlax-invoke-method srv "Free")
(setq left_list (vlax-safearray->list left_safe_arr)
      right_list (vlax-safearray->list right_safe_arr))
```

Диалог с пометкой элементов списка

В этом диалоговом окне (рис. 15.9) можно пометить элементы списка[1]. Удобно использовать в качестве единого центра для управления логическими переменными.

Библиотека — ruCheckListSrv.dll, объект `ruCheckListSrv.CheckList`, объявление методов см. в листинге 15.25, функцию для LISP — в листинге 15.26.

[1] Подробно этот пример рассмотрен в главе 47 книги Н. Полещука "AutoCAD 2004".

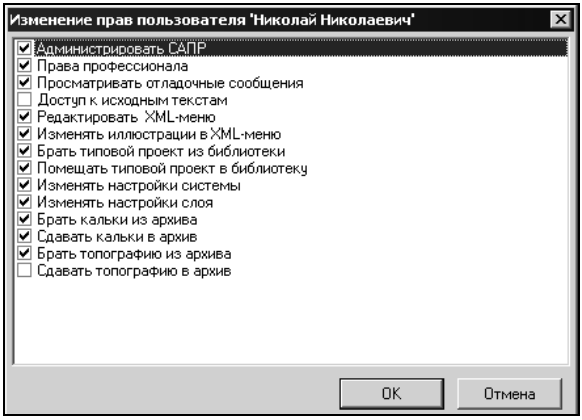


Рис. 15.9. Список с "галочками"

Листинг 15.25. Объявление объекта `ruCheckListSrv.CheckList`

```
type
  TCheckList = class(TAutoObject, ICheckList)
  protected
    function Execute: OleVariant; safecall;
    function Get_Items: IStrings; safecall;
    function Get_ItemsCount: Integer; safecall;
    procedure GetItem(Index: Integer; out Value: OleVariant); safecall;
    procedure AddItem(const CheckString: WideString; Enabled: WordBool);
      safecall;
    procedure Create; safecall;
    procedure Free; safecall;
    procedure Set_Caption(const Value: WideString); safecall;
    procedure Set_Items(const Value: IStrings); safecall;
  end;
```

Листинг 15.26. Функция `ru-dlg-show-check-list`

```
(defun ru-dlg-show-check-list  (dlg_caption lst_names lst_values
                read_only / count i lst_s result s srv is_edit)
;;; Вывод списка с отметками
;|
Аргументы:
dlg_caption - заголовок диалога
lst_names - список выводимых строк
lst_values - список в строковом виде пометок ("1" - помечено, "0" - не помечено)
read_only - пометки только для чтения?
Пример:
(ru-dlg-show-check-list "Проверка"
 (list "Переменная 1" "Переменная 2" "Переменная 3")
  (list "1" "0" "1")
  T
)
```

```
|;
 (if (setq srv (vlax-get-or-create-object "ru_ChkListSrv.ruCheckList"))
  (progn
    (setq is_edit (ru-conv-value-to-wordbool (not read_only))
    i 0 lst_s '())
    (repeat (length lst_names)
      (setq lst_s
       (cons (strcat (nth i lst_values) (nth i lst_names)) lst_s)
       i (1+ i))
   );_ end of repeat
    (setq lst_s (reverse lst_s))
    (vlax-invoke-method srv "Create")
    (vlax-put-property srv 'Caption dlg_caption)
    (foreach s lst_s
     (vlax-invoke-method srv "AddItem" s is_edit)
   );_ end of foreach
    (setq result (ru-conv-value-to-bool
     (vlax-invoke-method srv "Execute")))
    (if (= result :vlax-true)
      (if (not read_only)
       (progn
         (setq count (vlax-get-property srv 'ItemsCount) lst_values '()
          i 0)
        (repeat count
           (setq s "")
           (vlax-invoke-method srv "GetItem" i 's)
           (setq lst_values (cons s lst_values) i (1+ i))
       );_ end of repeat
        (setq result (reverse lst_values))
      );_ end of progn
       (setq result lst_values)
    );_ end of if
  );_ end of if
  (vlax-invoke-method srv "Free")
);_ end of progn
 (ru-msg-alert "Не удалось запустить ru_ChkListSrv.ruCheckList")
);_ end of if
 result
)
```

Диалоговое окно просмотра текстового файла

Просмотр (с факультативным редактированием) текстового файла (рис. 15.10) может осуществляться с помощью специального диалогового окна. Такое окно мы используем для вывода кратких справок о программах. Для облегчения собственной работы мы предусмотрели возможность редактирования текста. Во время отладки программы разработчик может править и сохранять текст, не разыскивая его в лабиринте каталогов и не загружая во внешний редактор. Обычный пользователь может только просматривать текст, а кнопку **Сохрани** просто не видит.

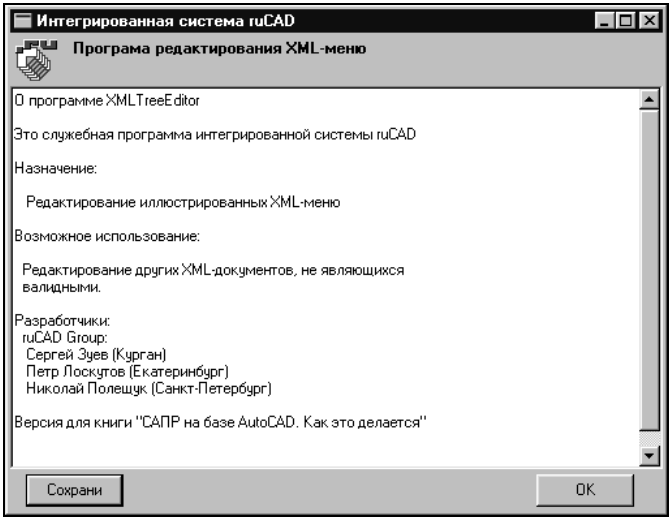


Рис. 15.10. Просмотр текстового файла

Библиотека — ruTxtViewSrv.dll, объект `ruTxtViewSrv.TxtFileViewer`, объявление методов см. в листинге 15.27, функцию для LISP — в листинге 15.28.

Листинг 15.27. Объявление объекта `ruTxtViewSrv.TxtFileViewer`

```
type
  TTxtFileViewer = class(TAutoObject, ITxtFileViewer)
  protected
    procedure Show(const FileName, Caption, Header: WideString;
      ReadOnly: WordBool); safecall;
end;
```

Листинг 15.28. Функция `ru-dlg-view-txt-file`

```
(defun ru-dlg-view-txt-file (file_name / srv)
;;; Просмотр файла file_name
(if (setq srv (vlax-get-or-create-object "ruTxtViewSrv.TxtFileViewer"))
  (vlax-invoke-method srv "Show" file_name
     (strcat "Просмотр " file_name) "Текстовый файл" 1)
  (alert "Не удалось запустить ruTxtViewSrv.TxtFileViewer ")
)
)
```

Визуальное редактирование координат полилинии

Редактор координат полилинии (см. *главу 10*, рис. 10.3) очень удобен не только для ввода заданного списка координат (в САПР это бывает редко, в основном при раз-

работке чертежей генпланов), но и для визуального контроля правильности трассировок линий, которые трудно отследить в рисунке. Этот же объект мы применяем при просмотре и редактировании пространственно-координированных данных, хранящихся в таблицах баз данных.

Библиотека — ruCoordEditSvr.dll, объект `ruCoordEditSvr.CoordEdit`, объявление методов см. в листинге 15.29, функции для LISP — в листингах 15.30—15.32.

Листинг 15.29. Объявление объекта `ruCoordEditSvr.CoordEdit`

```
type
  TCoordEdit = class(TAutoObject, ICoordEdit)
  protected
    function Edit2dCoords(const Caption, FileName: WideString;
      IsClosed: WordBool): OleVariant; safecall;
end;
```

Листинг 15.30. Функция `_ru-dlg-coord-edit-srv`

```
(defun _ru-dlg-coord-edit-srv ()
;;; Создание COM-сервера редактора координат
  (vlax-get-or-create-object "ruCoordEditSvr.CoordEdit")
)
```

Листинг 15.31. Функция `_ru-dlg-coord-edit-2d`

```
(defun _ru-dlg-coord-edit-2d (caption std_lst_coords
                  / is_closed lst_coords result srv tmp_file)
;| Низкоуровневая функция редактирования координат
Аргументы:
caption - заголовок диалогового окна
std_lst_coords - список координат вершин в стандартной форме
Пример:
(_ru-dlg-coord-edit-2d "Проба" (list (list
 (list 1000.0 6000.0)
 (list -566.0 1000.0)
 (list 340.0 444.0)
) T))
В этой функции используется передача списка координат через файл. Можно было бы
передавать координаты в виде массива, но в данном случае мы вынуждены работать
через файл для совместимости с некоторыми другими программами, не умеющими работать
с COM-объектами
|;
  (setq is_closed (cadr std_lst_coords)
    tmp_file (ru-file-tmp "coordedit.tmp"))
  (vl-file-delete tmp_file)
  (if (ru-list-coords-write-to-file tmp_file std_lst_coords)
    (if (setq srv (_ru-dlg-coord-edit-srv))
      (if (= (vlax-variant-value
               (vlax-invoke-method srv "edit2dCoords" caption tmp_file
                 (ru-conv-value-to-wordbool is_closed)
```

```
            );_ end of vlax-invoke-method
          );_ end of vlax-variant-value
           :vlax-true
        );_ end of =
       (if (setq lst_coords (ru-list-coords-read-from-file tmp_file))
         (setq result (list lst_coords is_closed))
      );_ end of if
    );_ end of if
  );_ end of if
 );_ end of if
  result
);_ end of defun
```

Листинг 15.32. Функция `ru-dlg-coord-edit`

```
(defun ru-dlg-coord-edit (dlg_header ent / edata lstdata newent
                          newlstdata newlstver std_lst_coords)
;| Диалоговый редактор полилинии
 Возвращает новый примитив или nil
Пример:
  (ru-dlg-coord-edit "Проба редактора" (entlast))
|;
  (if (setq std_lst_coords (ru-pline-list-vertex ent))
    (progn
      (setq edata (entget ent)
            lstdata (list
                      (assoc 8 edata) ;Слой
                      (assoc 62 edata);Color
                      (assoc 43 edata);Ширина
                      (assoc 6 edata) ;Ltype
                      (assoc 48 edata);Ltype scale
                      (assoc 38 edata);Elevation
                      (assoc 39 edata);Thickness
                      (assoc 70 edata);Closed
                      (assoc 370 edata);LW
                    );_ end of list
    );_ end of setq
     (if
       (setq newlstver
              (_ru-dlg-coord-edit-2d dlg_header std_lst_coords)
       );_ end of setq
;| Получили список новых вершин.Теперь можно перерисовать объект
|;
       (progn
         (if (setq newent (ru-pline-entmake (car newlstver)
                                (cadr newlstver) nil 0 0))
           (progn
               (setq newlstdata (entget newent))
               (while lstdata
                 (if (nth 0 lstdata)
```

```
                (setq newlstdata
                  (if (assoc (car (nth 0 lstdata)) newlstdata)
                      (subst (nth 0 lstdata)
                        (assoc (car (nth 0 lstdata)) newlstdata)
                         newlstdata
                       );_ end of subst
                       (append newlstdata (list (nth 0 lstdata)))
                   );_ end of if
                 );_ end of setq
               );_ end of if
                (setq lstdata (cdr lstdata))
             );_ end of while
              (entmod newlstdata)
              (entupd (cdr (assoc -1 newlstdata)))
              (entdel ent)
           );_ end of progn
            (princ "\nЛиния не изменена!")
         );_ end of if
       );_ end of progn
     );_ end of if
   );_ end of progn
    (princ "\nПримитив недопустимого типа!")
 );_ end of if
 newent
);_ end of defun
```

Диалоговое окно "Советы дня"

Еще одно диалоговое окно — окно полезных советов (рис. 15.11). Большинство западных программистов оснащают свои программы такими окнами, в то время как большинство наших пользователей эти диалоги считают бесполезными и не читают.

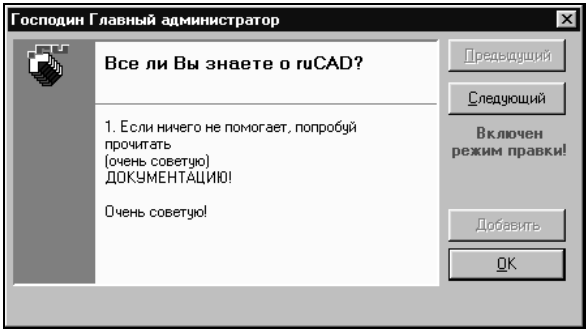


Рис. 15.11. Диалоговое окно полезных советов

Библиотека — ruTipsSrv.dll, объект `ruTipsSrv.ruTipsOfDay`, объявление методов см. в листинге 15.33, функцию для LISP — в листинге 15.34.

Листинг 15.33. Объявление объекта `ruTipsSrv.ruTipsOfDay`

```
type
  TruTipsOfDay = class(TAutoObject, IruTipsOfDay)
  protected
    procedure ShowTips(const FileName, DlgCaption, TipsHeader, IniFile,
      Section: WideString; CanEdit, CheckNextVisible: WordBool;
      NumberTip: Integer); safecall;
  end;
```

Листинг 15.34. Функция `ru-dlg-tips`

```
(defun ru-dlg-tips (dlg_caption tips_header file_name ini_file section
 can_edit / res_srv result srv)
;;; Вывод диалога "Советы дня"
;| Аргументы:
dlg_caption - заголовок диалогового окна
tips_header - заголовок советов
file_name - имя файла с советами
ini_file - имя INI-файла с настройками
section - секция INI-файла с настройками
can_edit - можно ли редактировать советы
Пример вызова:
(ru-dlg-tips
 (ru-user-long-name)
 "А знаете ли вы, что..."
 (strcat (ru-dirs-get-local-app-data) "TipsOfDay\\" name ".tips")
 (strcat (ru-dirs-get-local-app-data) "TipsOfDay\\tips.ini")
 name (ru-user-can-develop))
|;
  (if (not (findfile file_name))
    (vl-file-copy (ru-file-template "rucad.tips") file_name)
 );_ end of if
  (if (setq srv (vlax-get-or-create-object "ruTipsSrv.ruTipsOfDay"))
    (vlax-invoke-method srv "ShowTips" file_name dlg_caption
      tips_header ini_file section
      (ru-conv-value-to-wordbool can_edit) 0 -1
   );_ end of vlax-invoke-method
    (ru-msg-alert "Не удалось запустить SingleListDlg")
 );_ end of if
);_ end of defun
```

Файлы с советами располагаются в папке %ruLocalAppDataDir%\TipsOfDay\ и обычно имеют расширение tips. Это простые текстовые файлы, в которых каждая строка содержит один совет. Для перевода строки в окне используется разделитель |, например:

```
Если ничего не помогает, попробуй |прочитать |ДОКУМЕНТАЦИЮ!
```

Если диалоговое окно вызвал пользователь с правами разработчика, появляется возможность редактировать советы прямо в диалоговом окне, не разыскивая файл.

Выбор файла из виртуального дерева

Бывает полезно организовать быстрый выбор файла из виртуального дерева (рис. 15.12), имитирующего меню. Пользователь может отредактировать файл и выбрать требуемый. О возможностях применения можно догадаться из рисунка — выбор файла с типовым текстом и, очевидно, последующая вставка текста в рисунок.

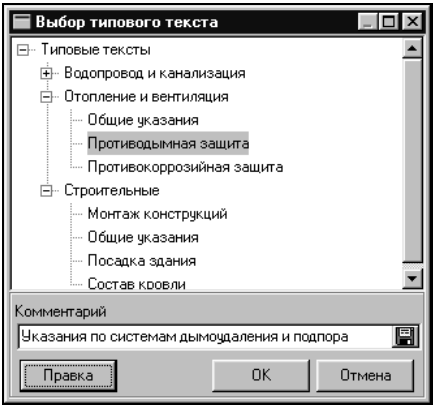


Рис. 15.12. Выбор документа из дерева

Библиотека — ruTreeDirSrv.dll, объект `ruTreeDirSrv.SelectFile`, объявление методов см. в листинге 15.35, функцию для LISP — в листинге 15.36.

Листинг 15.35. Объявление объекта `ruTreeDirSrv.SelectFile`

```
type
  TSelectFile = class(TAutoObject, ISelectFile)
  protected
    function Execute(const Header, RootDir, FileExt: WideString;
      EditMode: WordBool): OleVariant; safecall;
end;
```

Листинг 15.36. Функция `ru-dlg-file-select-in-tree`

```
(defun ru-dlg-file-select-in-tree (dlg_caption root_dir file_ext
                    is_can_edit / srv)
;;; Выбор файла из дерева
;|
Пример:
(ru-dlg-file-select-in-tree "Выбор файла" (ru-dirs-get-lsp-source)
   ".lsp" T)
|;
  (if (setq srv (vlax-get-or-create-object "ruTreeDirSrv.SelectFile"))
   (vlax-variant-value (vlax-invoke-method srv "Execute" dlg_caption
      root_dir file_ext (ru-conv-value-to-wordbool is_can_edit))
  )
```

```
  (ru-msg-alert "Не удалось запустить ruTreeDirSrv.SelectFile")
 );_ end of if
)
```

Системные папки Windows

Теперь разработаем несколько "тихих" функций, возвращающих различные свойства операционной системы, в основном каталоги системных папок. Нельзя в программах использовать фиксированные имена каталогов из так называемого "пространства имен" (namespace) Windows. Эти каталоги могут иметь разные имена. Даже если вы всегда имели дело с каталогом C:\Program Files, это не означает, что на всех компьютерах этой *папке* назначен именно такой *каталог*.

Для вычисления каталогов системных папок создадим библиотеку — ruSysInfoSvr.dll, объект `ruSysInfoSvr.SystemInfo`, объявление методов см. в листинге 15.37.

Листинг 15.37. Объявление объекта `ruSysInfoSvr.SystemInfo`

```
type
  TSystemInfo = class(TAutoObject, ISystemInfo)
  protected
    function GetAppdataFolder: WideString; safecall;
    function GetCommonAppdataFolder: WideString; safecall;
    function GetCommonFilesFolder: WideString; safecall;
    function GetCommonProgramsFolder: WideString; safecall;
    function GetCurrentFolder: WideString; safecall;
    function GetDesktopFolder: WideString; safecall;
    function GetDomainName: WideString; safecall;
    function GetLocalComputerName: WideString; safecall;
    function GetLocalUserName: WideString; safecall;
    function GetNethoodFolder: WideString; safecall;
    function GetPersonalFolder: WideString; safecall;
    function GetProgramFilesFolder: WideString; safecall;
    function GetProgramsFolder: WideString; safecall;
    function GetRecentFolder: WideString; safecall;
    function GetRegisteredCompany: WideString; safecall;
    function GetRegisteredOwner: WideString; safecall;
    function GetTemplatesFolder: WideString; safecall;
    function GetWindowsFolder: WideString; safecall;
    function GetWindowsSystemFolder: WideString; safecall;
    function GetWindowsTempFolder: WideString; safecall;
end;
```

У этого объекта имеются несколько функций, возвращающих имена системных папок. На всякий случай мы включили и функции, возвращающие имена компьютера, пользователя и владельца лицензии. Все это записано в реестре и мы могли бы добраться до этих параметров и из Visual LISP, но мы используем единый объект автоматизации для всех своих программ, чтобы сократить вероятность собственных ошибок. Для практического использования сначала разработаем низкоуровневую функцию (листинг 15.38).

Листинг 15.38. Функция _ru-sys-info

```
(defun _ru-sys-info (method_name / srv)
;; Возвращает системную информацию с заданным по имени методом
;|
Допустимые имена методов приведены в примере.
Пример:
(list
  (_ru-sys-info "GetAppdataFolder")
  (_ru-sys-info "GetCommonAppdataFolder")
  (_ru-sys-info "GetCommonFilesFolder")
  (_ru-sys-info "GetCommonProgramsFolder")
  (_ru-sys-info "GetCurrentFolder")
  (_ru-sys-info "GetDesktopFolder")
  (_ru-sys-info "GetDomainName")
  (_ru-sys-info "GetLocalComputerName")
  (_ru-sys-info "GetLocalUserName")
  (_ru-sys-info "GetNethoodFolder")
  (_ru-sys-info "GetPersonalFolder")
  (_ru-sys-info "GetProgramFilesFolder")
  (_ru-sys-info "GetProgramsFolder")
  (_ru-sys-info "GetRecentFolder")
  (_ru-sys-info "GetRegisteredCompany")
  (_ru-sys-info "GetRegisteredOwner")
  (_ru-sys-info "GetTemplatesFolder")
  (_ru-sys-info "GetWindowsFolder")
  (_ru-sys-info "GetWindowsSystemFolder")
  (_ru-sys-info "GetWindowsTempFolder")
);_ end of list
|;
 (if (setq srv (vlax-get-or-create-object "ruSysInfo.SystemInfo"))
   (vlax-invoke-method srv method_name)
    (alert "Не могу запустить ruSysInfo.SystemInfo")
 );_ end of if
);_ end of defun
```

Для того чтобы не ошибиться при передаче имени метода и не путаться с номерами методов (так сделано в библиотеке DOSLib), мы напишем несколько функций для вызова каждого из методов. Пример одной из них см. в листинге 15.39.

Листинг 15.39. Функция ru-dirs-get-sys-program-files-folder

```
(defun ru-dirs-get-sys-program-files-folder ()
  (_ru-sys-info "GetProgramFilesFolder")
)
```

Системные папки ruCAD

В нашей системе есть свои "системные" каталоги. В *главе 12* мы разработали несколько функций, определяющих расположение таких каталогов путем прямого чте-

ния реестра. Теперь мы разработали (и еще будем разрабатывать) много Delphi-приложений, использующих эти же данные. Наступила пора передать эти операции в единый центр — объект автоматизации. Создадим библиотеку — ruIniRegSrv.dll, объект `ruIniRegSrv.RegIni`, объявление методов см. в листинге 15.40.

Листинг 15.40. Объявление объекта `ruIniRegSrv.RegIni`

```
type
  TRegIni = class(TAutoObject, IRegIni)
  protected
    function Get_AllUsersDir: WideString; safecall;
    function Get_AppDataDir: WideString; safecall;
    function Get_CurrentUserDir: WideString; safecall;
    function Get_HtmlDir: WideString; safecall;
    function Get_LocalAcadAllVersionDir: WideString; safecall;
    function Get_LocalAppDataDir: WideString; safecall;
    function Get_LocalSettingsDir: WideString; safecall;
    function Get_LspSourceDir: WideString; safecall;
    function Get_RootDir: WideString; safecall;
    function Get_XmlImagesDir: WideString; safecall;
    function Get_XmlMenuDir: WideString; safecall;
    procedure Set_AllUsersDir(const Value: WideString); safecall;
    procedure Set_AppDataDir(const Value: WideString); safecall;
    procedure Set_CurrentUserDir(const Value: WideString); safecall;
    procedure Set_HtmlDir(const Value: WideString); safecall;
    procedure Set_LocalAcadAllVersionDir(const Value: WideString); safecall;
    procedure Set_LocalAppDataDir(const Value: WideString); safecall;
    procedure Set_LocalSettingsDir(const Value: WideString); safecall;
    procedure Set_LspSourceDir(const Value: WideString); safecall;
    procedure Set_RootDir(const Value: WideString); safecall;
    procedure Set_XmlImagesDir(const Value: WideString); safecall;
    procedure Set_XmlMenuDir(const Value: WideString); safecall;
    function Get_ruCADVersion: WideString; safecall;
    procedure Set_ruCADVersion(const Value: WideString); safecall;
end;
```

В рассматриваемом случае мы привели заведомо неверное объявление. Все эти процедуры и функции действительно имеются в объекте, но они недоступны из Visual LISP потому, что мы не просто объявляли метод при разработке библиотеки типов, а создавали свойства для чтения и записи. Указанные методы используются внутри свойств, поэтому настоящим является объявление, приведенное в листинге 15.41.

Листинг 15.41. Фрагмент файла ruIniRegSrv_TLB.pas

```
IRegIniDisp = dispinterface
    ['{80716742-D54E-11D7-8CF1-CF934FB0FE5E}']
    property RootDir: WideString dispid 1;
    property AllUsersDir: WideString dispid 2;
    property AppDataDir: WideString dispid 3;
    property CurrentUserDir: WideString dispid 4;
```

```
    property HtmlDir: WideString dispid 5;
    property XmlMenuDir: WideString dispid 6;
    property XmlImagesDir: WideString dispid 7;
    property LocalAcadAllVersionDir: WideString dispid 8;
    property LocalAppDataDir: WideString dispid 9;
    property LocalSettingsDir: WideString dispid 10;
    property LspSourceDir: WideString dispid 11;
    property ruCADVersion: WideString dispid 12;
  end;
```

LISP-функция для чтения свойств приведена в листинге 15.42, а для записи свойств — в листинге 15.43.

Листинг 15.42. Функция _ru-reg-get-registered

```
(defun _ru-reg-get-registered (property_name / srv)
;|
Чтение свойства по имени из реестра
Имена свойств должны соответствовать примеру
Пример:
(list
    (_ru-reg-get-registered "AllUsersDir")
    (_ru-reg-get-registered "AppDataDir")
    (_ru-reg-get-registered "CurrentUserDir")
    (_ru-reg-get-registered "HtmlDir")
    (_ru-reg-get-registered "LocalAcadAllVersionDir")
    (_ru-reg-get-registered "LocalAppDataDir")
    (_ru-reg-get-registered "LocalSettingsDir")
    (_ru-reg-get-registered "LspSourceDir")
    (_ru-reg-get-registered "RootDir")
    (_ru-reg-get-registered "XmlImagesDir")
    (_ru-reg-get-registered "XmlMenuDir")
    (_ru-reg-get-registered "ruCADVersion")
)
|;
  (if (setq srv (vlax-get-or-create-object "ruIniRegSrv.RegIni"))
    (vlax-get-property srv property_name)
    (alert "Не могу запустить ruIniRegSrv.RegIni")
 )
)
```

Листинг 15.43. Функция _ru-reg-set-registered

```
(defun _ru-reg-set-registered (property_name value / srv)
;|
Запись в реестр ruCAD значения свойства
|;
  (if (setq srv (vlax-get-or-create-object "ruIniRegSrv.RegIni"))
    (vlax-put-property srv property_name value)
    (alert "Не могу запустить ruIniRegSrv.RegIni")
 )
)
```

Для облегчения практического применения мы напишем несколько функций для чтения и записи каждого свойства. Два примера приведены в листингах 15.44 и 15.45.

Листинг 15.44. Функция `ru-reg-get-root-dir`

```
(defun ru-reg-get-root-dir ()
  (_ru-reg-get-registered "RootDir")
)
```

Листинг 15.45. Функция `ru-reg-set-root-dir`

```
(defun ru-reg-set-root-dir (value)
  (_ru-reg-set-registered "RootDir" value)
)
```

Теперь нам необходимо изменить ранее написанные функции. Например, функцию определения корневого каталога системы можно переписать так, как показано в листинге 15.46.

Листинг 15.46. Измененная функция `ru-dirs-get-root`

```
(defun ru-dirs-get-root()
;;; Новая функция, определяющая корневой каталог системы
;;; Для ускорения вводим глобальную переменную
  (cond (*ru_root_dir*)
   (T (setq *ru_root_dir* (ru-reg-get-root-dir)))
 );_ end of cond
);_ defun
```

По такой же схеме переписываем и все остальные функции, ранее читавшие данные из реестра:

- `ru-dirs-get-all-users`;
- `ru-dirs-get-app-data`;
- `ru-dirs-get-current-user`;
- `ru-dirs-get-html`;
- `ru-dirs-get-layers-class`;
- `ru-dirs-get-local-acad-all-version`;
- `ru-dirs-get-local-app-data`;
- `ru-dirs-get-local-settings`;
- `ru-dirs-get-lsp-source`.

Диалоговые окна в EXE-файлах

Мы разобрали много примеров создания COM-серверов на Delphi с использованием их в LISP-приложениях. Но это не единственная возможность сочетания Delphi и AutoCAD. Еще до появления COM-технологий мы использовали в LISP-функциях диалоговые окна, изготовленные в виде самостоятельных приложений. Такая технология с успехом применялась еще с системой AutoCAD R10. Разумеется, приложения разрабатывались не на Delphi, а на Turbo Pascal и Clipper. Многие задачи легко

решаются в самостоятельных приложениях, но иногда такое приложение должно получить от AutoCAD набор каких-либо исходных данных, причем разных типов, и вернуть в AutoCAD результаты своей работы. Часто такие приложения разрабатываются не специально (или не только) для AutoCAD.

В подобных случаях применяется следующая схема работы:

- Основная программа пишется на LISP. Из LISP-программы запускается в модальном режиме внешнее приложение, а LISP-программа ждет завершения его работы. Специально написанное внешнее приложение может имитировать функцию LISP.
- После завершения внешнего приложения LISP-программа продолжает работу, (возможно, получив результаты работы внешнего приложения через файл).

Зачем нужно так делать

Действительно, зачем? Мы только что рассмотрели массу примеров, где никаких файлов для обмена данными не используется. Все прекрасно работает. Однако далеко не все разработчики безоговорочно следуют "решениям Microsoft". Некоторые не хотят, а некоторые не могут. Нам встречаются приложения, результаты работы которых просто необходимо использовать в AutoCAD (в основном это расчетные программы). При всей относительной простоте разработки COM-серверов (в подходящей среде) изготовить автономное приложение все-таки проще. Бояться обмена данными посредством файла не нужно. Многие UNIX-системы основаны именно на таком обмене данных и работают, мягко говоря, нисколько не хуже WIN-приложений, основанных на хитроумных способах обмена данными через память.

Теперь рассмотрим ситуацию, в которой COM не поможет. Очень распространенная задача — в диалоговом окне нужно иметь "кнопку", при нажатии которой пользователь должен получить какие-то данные из системы AutoCAD (указать точку, выбрать примитив, изменить вид, нарисовать объект). Простейший пример был рассмотрен в *главе 10* при разработке функции `ru-dlg-dcl-get-string` (см. листинг 10.40). В диалоговом окне (см. рис. 10.4) имеется кнопка, при щелчке по которой можно выбрать текст указанием в рисунке, причем указанием на любой объект, *похожий* на текст (размер, атрибут, многострочный или обычный текст). Извлечение строки из объекта производится функцией `ru-get-txt-from-dwg`. Диалоговое окно по современным меркам имеет довольно убогий вид. Несколько лет мы его совершенствовали, но улучшить его с использованием DCL вряд ли возможно. Код функции также довольно объемен и не очень прост.

Такое диалоговое окно мы легко можем сконструировать в Delphi и создать еще один COM-сервер. Но как сделать, чтобы в нужный момент это диалоговое окно скрылось с экрана, запустилась бы функция `ru-get-txt-from-dwg`, пользователь сделал бы выбор (или отказался от выбора), а результат возвратился бы в диалог в виде строки?

Конечно, можно изловчиться, отыскать объект `AcadApplication`, свернуть диалоговое окно и выполнить какой-то метод AutoCAD. В лучшем случае мы могли бы выполнить `Utility.GetEntity`, а потом анализировать объект в Delphi, что намного сложнее, чем в LISP. Но ведь AutoCAD в этот момент занят выполнением нашего приложения, и не будет откликаться на посторонние запросы! Не беремся категориче-

ски утверждать, что решение такой задачи вообще невозможно. Все может быть, но в любом случае решение будет очень непростым и очень ненадежным. Но такую задачу очень легко решить с помощью обычного EXE-файла, и мы это сделаем немного позже.

Как запустить внешнее приложение в модальном режиме

К сожалению, в LISP нет такой необходимой функции, но мы уже ее разработали именно с такой целью, да еще в нескольких вариантах (последний приведен в листинге 15.8).

Как передать параметры

Передать параметры внешнему приложению можно с помощью командной строки. Однако параметров может быть много. Самыми известными примерами программ с множеством параметров, команд и опций являются архиваторы. Писать такие программы не очень просто, значительную их часть занимает разбор параметров. После многих экспериментов мы пришли к выводу, что лучшим способом является передача в качестве единственного параметра имени INI-файла, в котором могут быть заданы любые постоянные или временные переменные, требующиеся для работы приложения.

Как получить результаты

Результаты работы приложения лучше также получать через файл. Это может быть тот же стартовый INI-файл. Если результатом является большое количество данных, они могут записываться в файл любого удобного формата. Не следует забывать, что результаты могут иметь вид сгенерированного LISP-выражения или целой программы.

Функция ввода строки

Теперь реализуем изложенную концепцию на примере функции ввода строки. Delphi-приложение такого класса сделать очень просто. Внешний вид диалогового окна во время работы приведен на рис. 15.13, модуль главной формы — в листинге 15.47.

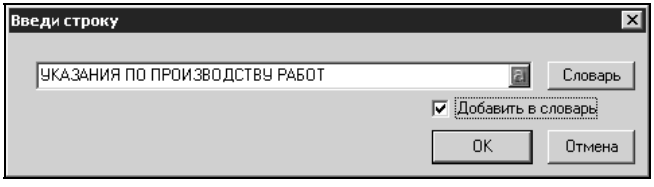


Рис. 15.13. Диалоговое окно ввода строки

Листинг 15.47. Файл ru_GetStrFrm.pas

```
unit ru_GetStrFrm;

interface

uses Windows, SysUtils, Classes, Graphics, Forms, Controls, StdCtrls,
  Buttons, ExtCtrls, Mask, JvToolEdit, JvComponent, ruSingleListSrv_TLB;

type
  TfrmGestAcadStrDlg = class(TForm)
    btnOk: TButton;
    btnCancel: TButton;
    JvComboEdit1: TJvComboEdit;
    CheckBox1: TCheckBox;
    ButtonDic: TButton;
    LabelError: TLabel;
    procedure ButtonDicClick(Sender: TObject);
    procedure btnOkClick(Sender: TObject);
    procedure btnCancelClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure JvComboEdit1ButtonClick(Sender: TObject);
    procedure JvComboEdit1Change(Sender: TObject);
  private
    { Private declarations }
{
Используем для выбора из словаря только что разработанный COM-сервер
}
    SingleListDlg: ISingleListDlg;
  public
    { Public declarations }
  end;
var
  frmGestAcadStrDlg: TfrmGestAcadStrDlg;

var
  DisableEmpty: boolean;
  IniFileName, DicFileName: string;
  BakLst, DicLst: TStringList;
  DialResult: Integer;

implementation
uses
  ComObj, Dialogs, abIniFiles, ruUtils;

{$R *.DFM}
procedure TfrmGestAcadStrDlg.FormCreate(Sender: TObject);
var
  Ini: TIniFile;
Begin
{
Имя INI-файла - единственный, но обязательный параметр приложения
}
```

```
  IniFileName := ruGetParameter();
  if IniFileName <> '' then
  begin
    SingleListDlg := CreateComObject(CLASS_SingleListDlg) as ISingleListDlg;
    DicLst := TStringList.Create;
    BakLst := TStringList.Create;
{
Чтение "задания" из файла
}
    ini := TIniFile.Create(IniFileName);
    Caption := ini.ReadString('Start', 'Caption', 'Ввод строки');
    JvComboEdit1.Text := ini.ReadString('Start', 'DefString', '');
    DisableEmpty := ini.ReadBool('Start', 'DisableEmpty', True);
    DicFileName := ini.ReadString('Start', 'DicFileName', DicFileName);
    ini.free;
  end;
end;

procedure TfrmGestAcadStrDlg.ButtonDicClick(Sender: TObject);
{
Выбор строки из словаря. Словарь отображается в знакомом нам по этой главе
диалоговом окне выбора из списка. Мы используем его в виде объекта автоматизации
}
var
  i: integer;
  ResDial: OleVariant;
begin
  DicLst.LoadFromFile(DicFileName);
  BakLst.LoadFromFile(DicFileName);
  SingleListDlg.Create;
  SingleListDlg.DlgCaption := 'Выбор строки из словаря';
  SingleListDlg.ListCaption := 'Словарь';
  for i := 0 to DicLst.Count - 1 do
  begin
    SingleListDlg.AddStringToList(DicLst[i]);
  end;
  ResDial := SingleListDlg.Execute;

  if ResDial then
  begin
    JvComboEdit1.Text := SingleListDlg.ResultString;
  end;
  SingleListDlg.Free;

end;

procedure TfrmGestAcadStrDlg.btnOkClick(Sender: TObject);
var
  index: integer;
begin
{
При щелчке по кнопке OK приложение закрывается. Значение переменной DialResult := 1
перед закрытием формы
}
```

```
  if (CheckBox1.Checked and (JvComboEdit1.Text <> '')) then
  begin
    index := BakLst.IndexOf(JvComboEdit1.Text);
    if (index = -1) then
    begin
      BakLst.Insert(0, JvComboEdit1.Text);
      BakLst.SaveToFile(DicFileName);
    end;
  end;
  DialResult := 1;
  Close;
end;

procedure TfrmGestAcadStrDlg.btnCancelClick(Sender: TObject);
begin
{
При щелчке по кнопке Cancel приложение закрывается. Значение переменной DialResult
:= 2 перед закрытием формы
}
  DialResult := 2;
  Close;
end;

procedure TfrmGestAcadStrDlg.JvComboEdit1ButtonClick(Sender: TObject);
begin
{
При щелчке по кнопке с изображением логотипа AutoCAD приложение закрывается.
Значение переменной DialResult := 3 перед закрытием формы
}
  DialResult := 3;
  Close;
end;

procedure TfrmGestAcadStrDlg.JvComboEdit1Change(Sender: TObject);
begin
  if DisableEmpty then
  begin
    btnOk.Enabled := JvComboEdit1.Text <> '';
  end
  else
    btnOk.Enabled := True;
  LabelError.Visible := (not btnOk.Enabled);
end;

procedure TfrmGestAcadStrDlg.FormClose(Sender: TObject;
  var Action: TCloseAction);
var
  Ini: TIniFile;
begin
{
При закрытии формы текущее состояние записывается в файл результатов. Обратите
внимание на то, что записано значение переменной DialResult, которое при выходе
```

```
по кнопке OK будет равно 1, при выходе по кнопке Cancel будет равно 2, а при выходе
по кнопке "Взять в AutoCAD" будет равно 3
}
  ini := TIniFile.Create(IniFileName);
  ini.WriteString('Result', 'String', JvComboEdit1.Text);
  ini.WriteInteger('Result', 'ResultDial', DialResult);
  ini.free;
  DicLst.Free;
  BakLst.Free;
end;
end.
```

Как видите, код Delphi-приложения очень прост. Теперь разработаем несложную функцию, управляющую этим приложением (листинг 15.48).

Листинг 15.48. Функция `ru-dlg-get-string`

```
(defun ru-dlg-get-string (caption default disable_empty /
 command_line continue ini_file reason result user_dic_file tmp_str)
;|
Диалог ввода строки. Аргументы:
caption  - заголовок диалогового окна
default - строка по умолчанию
disable_empty - запрещен возврат пустой строки?
Примеры:
 (ru-dlg-get-string "Ввод необязательной строки" "Старая строка"  NIL)
 (ru-dlg-get-string "Ввод обязательной строки" "Старая строка"  T)
|;
  (setq ini_file (strcat (ru-dirs-get-local-app-data)
                 "GetString\\GetString.ini")
        user_dic_file (ru-file-current-user "userstr.txt")
        command_line  (strcat (ru-file-bin "ru_GetStr.exe") " " ini_file)
        result        default
        continue      t
 );_ end of setq
  (if (not (findfile user_dic_file))
    (vl-file-copy
      (ru-file-template "userstr.txt")
      user_dic_file
   );_ end ofvl-file-copy
 );_ end ofif
  (vl-file-delete ini_file)
  (ru-ini-write ini_file "Start" "Caption" caption)
  (ru-ini-write ini_file "Start" "DicFileName" user_dic_file)
  (ru-ini-write ini_file "Start" "DisableEmpty"
     (ru-conv-value-to-wordbool disable_empty))
;;; Запуск диалога в цикле
  (while continue
    (ru-ini-write ini_file "Start" "DefString" result)
    (ru-app-run command_line t)
```

```
    (setq result (ru-ini-read ini_file "Result" "String" "")
          reason (atoi
                   (ru-ini-read ini_file "Result" "ResultDial" "1")
                 );_ end of ru-conv-str-to-bool
   );_ end of setq
    (cond
      ((= reason 3)
      ;;Выбор из Acad
       (if (setq tmp_str
                  (ru-get-txt-from-dwg
                    "Выбери надпись из рисунка"
                  );_ end of ru-get-txt-from-dwg
           );_ end of setq
          (setq result tmp_str)
       );_ end of if
      )
       ((= reason 1)
       ;;OK
        (setq continue nil)
       ;; вернуть строку
      )
       (T
       ;; Отмена
        (setq continue nil
              result nil
       );_ end of setq
      )
   );_ end of cond
 );_ end of while
  result
);_ end of defun
```

Реализация функции на LISP также очень проста. "Задание" записывается в файл, в "модальном" режиме запускается приложение, после завершения приложения анализируется результат, если в переменной `ResultDial` INI-файла значение 3, запускается функция `ru-get-txt-from-dwg`, результат ее работы записывается в файл и вновь запускается приложение. Если в `ResultDial` содержатся 1 или 2, происходит выход из цикла запусков EXE-файла.

По аналогичной схеме мы разработаем еще немало функций, но подробно разбирать их уже не будем.

Резюме

В этой главе мы рассмотрели, как создавать свои функции для LISP с использованием среды разработки Borland Delphi и COM-технологий. Примерно так же, а может быть и проще, это можно делать с использованием Visual Basic (не VBA) и Visual C++. Хорошим примером использования COM является программа ACADVar, которую мы несколько раз упоминаем в книге и которую вы сможете найти на при-

лагаемом компакт-диске. Автор этой программы Алексей Васильченко не только весьма разумно использует Visual Basic в сочетании с Visual LISP для решения собственных задач, но и прекрасно документировал разработанные им объекты автоматизации, что совсем редко встречается в бесплатных программах.

Для профессионалов COM-технологий, возмущенных изложением материалов этой главы, мы рекомендуем еще раз прочитать начало главы, а остальным читателям советуем хорошо изучить COM по специализированной литературе для осмысленного использования таких решений.

В ходе разработки системы ruCAD приведенные решения будут не один раз изменяться — как для Delphi-приложений, так и для LISP-функций. Пока система не распространяется, допустимо даже изменение интерфейсов, но, как только наши библиотеки будут использоваться сторонними разработчиками, изменение интерфейсов будет недопустимым. Нам, при необходимости, придется создавать новые варианты COM-объектов, оставляя в неприкосновенности объявления старых, хотя реализацию объектов мы можем пересматривать.

ГЛАВА 16

Работа с XML

В этой книге мы много раз упоминали про дерево меню, на использовании которого будут основываться многие наши программы. В *главе 8* мы даже кратко познакомили читателей с синтаксисом такого меню. Теперь нам предстоит реализовать задуманное. Прежде всего кратко познакомимся с основами языка XML.

Минимальные сведения по XML

Расширяемый язык разметки XML (Extensible Markup Language) был разработан организацией World Wide Web Consortium (W3C) для размещения информации во Всемирной паутине (World Wide Web). Спецификация языка была опубликована в 1998 году. В этой книге мы вообще не будем касаться применения XML по его основному назначению и связанных с этими аспектами технологий. Для их описания требуется примерно столько же книг, сколько уже написано про AutoCAD. Мы затронем только маленький краешек XML, в интересующем нас направлении.

Читатели книги "AutoCAD 2002" (серия "В подлиннике") и пользователи системы BestIA знакомы с синтаксисом иллюстрированных меню, применявшихся в этой системе. Мы разрабатывали эти меню в те же годы, когда рабочая группа W3C продумывала основы XML. Синтаксис наших меню тех лет напоминал еще несуществующий XML и обладал большим недостатком — его могла читать только наша программа. Как только XML был разработан и опубликован, нам с первого взгляда стало понятно, что именно этот язык решает наши проблемы, причем стандартными для всего мира средствами. Нас не надо было убеждать в преимуществах XML, а именно такие уверения составляют значительную часть книг по XML.

Язык XML напоминает знакомый многим язык гипертекстовой разметки HTML. Но, в отличие от HTML, в котором используется большой, но фиксированный набор элементов (заголовки, абзацы, таблицы, списки и т. п.), в XML можно создавать собственные элементы, присваивать им имена и собственные атрибуты. Следовательно, XML можно использовать для описания любого документа, от простой ведомости документов до большого рисунка AutoCAD. Это поняли многие разработчики и стали использовать XML совсем не по первоначальному назначению. В AutoCAD 2004 в формате XML описаны инструментальные палитры (Tool Palettes); AutoCAD 2002 сохраняет и вставляет блоки в формате DesignXML (вариант XML для хранения графических данных); разработана спецификация LandXML, позволяющая

хранить любую геоинформацию; многие программы хранят в формате XML свои настройки.

Мы будем применять язык XML для:

- описания иллюстрированных меню;
- хранения данных и опций программ с возможностью удобного выбора;
- хранения и редактирования табличных данных.

Знакомство с XML мы начнем с реального меню строительной части (листинг 16.1).

Листинг 16.1. Файл ar.xml

```
<?xml version='1.0' encoding='windows-1251'?>
<Меню name='Архитектурно-строительные'>
  <Оси name='Оси здания' image='ar\axis.gif'
       macro='(ru-app-load "ru_ar_laying_axis")'>
  Рисование координационных осей здания со множеством опций:
  1. Задание постоянного и переменного шага
  2. Количество осей
  3. Направление осей
  4. Автоматическая нумерация
  </Оси>
  <Планы include='ar_plan.xml'/>
  <Разрезы include='ar_razr.xml'/>
  <Фасады include='ar_fasad.xml'/>
   <Конструкции>
    <КЖ include='ar_kgb.xml'/>
    <КМ include='ar_km.xml'/>
    <КД include='ar_kd.xml'/>
     <item name='Соединения конструкций' include='ar_soed.xml'/>
   </Конструкции>
</Меню>
```

Давайте сразу разберемся с терминологией.

Строка `<?xml version='1.0' encoding='windows-1251'?>` называется *объявлением документа*. Объявление можно не вводить в состав документа, но тогда весь документ должен быть написан латинскими символами, иначе многие программы "споткнутся" на первой же русской букве. В нашем случае указано, что документ является документом XML версии 1.0 (других пока нет) и имеет кодировку Windows-1251.

Строка `<Меню name='Архитектурно-строительные'>` является *тегом* начала *элемента* с именем `Меню`, а строка `</Меню>` — конечным тегом элемента `Меню`. Этот элемент является *корневым элементом* документа. *Наличие одного (и только одного) корневого элемента является обязательным правилом XML. Имя элемента в начальном теге должно в точности соответствовать имени элемента в конечном теге, включая регистр символов.*

Внутри корневого (и любого другого) элемента содержатся другие элементы и символьные данные или ничто не содержится. Элементы организованы в древовидную

иерархическую структуру. Документ может содержать элементы с одинаковыми именами. При назначении имен необходимо придерживаться следующих правил:

- имя элемента должно начинаться с буквы или символа подчеркивания, после чего могут идти буквы, цифры, символы точки, тире или подчеркивания;
- имена, начинающиеся с префикса `xml`, зарезервированы для стандартных имен;
- двоеточие в имени зарезервировано для задания пространства имен; использовать в своих документах пространства имен мы не будем, не будем и применять двоеточие.

В нашем документе большинство элементов начинаются с символа < и заканчиваются символами />. Это краткая запись, указывающая на отсутствие *текста элемента*. Информация у этих элементов содержится только в атрибутах.

Элементы должны быть вложены упорядоченным образом. Если элемент начинается внутри другого элемента, то он должен и заканчиваться внутри этого элемента.

Элемент `Оси` имеет текст (выделен в листинге 16.1 курсивом). Другие элементы не имеют текста. Текст обычно предназначен для отображения на страницах в Интернете. Мы будем использовать текст элемента в других целях.

Кроме имени и текста элементы могут иметь *атрибуты*. В нашей системе атрибутам придается особая роль. Описание атрибута представляет пару `имя='значение атрибута'`. *Имена атрибутов должны соответствовать таким же правилам, как и у имен элементов. Имя атрибута должно только один раз присутствовать в одном элементе*.

Элемент `Меню` в нашем примере имеет атрибут `name` со значением `Архитектурно-строительные`, элемент `Оси` также имеет атрибут `name` со значением `Оси здания`, атрибут `image` со значением `ar\axis.gif` и атрибут `macro` со значением `'(ru-app-load "ru_ar_laying_axis")'`.

Элемент `Конструкции` не имеет ни текста, ни атрибутов, но включает другие элементы, а элемент `КЖ` имеет только атрибут `include` со значением `'ar_kgb.xml'`.

Особое значение в XML имеют кавычки. Строка, заключенная в кавычки, называется *литералом*. В нашем документе литералами являются значения атрибутов. *Литерал не может содержать внутри себя тот же символ кавычки, которым он ограничен с двух сторон*. Обычно в XML в качестве ограничителей используются двойные кавычки, но т. к. в наших документах часто будут встречаться LISP-выражения, внутри которых требуется использование двойных кавычек, мы будем ограничивать литералы одинарными кавычками. А вот одинарную кавычку, используемую в LISP, нам придется заменять на функции `quote`, `function` и `list`. *В литералах не могут содержаться символы < или &* (если это не ссылка на символ). Именно поэтому мы договаривались об исключении этих символов из имен наших функций.

На этом изучение языка XML можно приостановить. Для наших целей полученных знаний пока хватит.

Если мы загрузим документ, приведенный в листинге 16.1, в Internet Explorer, то увидим этот документ в таком виде, как показано на рисунке (рис. 16.1).

Не очень красиво, но достаточно понятно. По крайней мере видно, что Internet Explorer понимает то, что мы написали. Браузеры Opera и Mozilla отобразят только *тексты* элементов (рис. 16.2), но не отобразят атрибуты, а у нас единственный эле-

мент Оси имеет текст. То, что три "независимых эксперта" от конкурирующих фирм "признали" наш документ, говорит о том, что он является *корректно сформированным*[1] (well-formatted, по терминологии W3C) документом. Различия в отображении объясняются различными *таблицами стилей*, примененными браузерами для отображения нашего документа. На таблицах стилей в этой книге мы не будем останавливаться, только заметим для себя, что и Internet Explorer не получал от нас письменных указаний заменять одинарные кавычки на двойные и отображать значения атрибутов полужирным шрифтом.

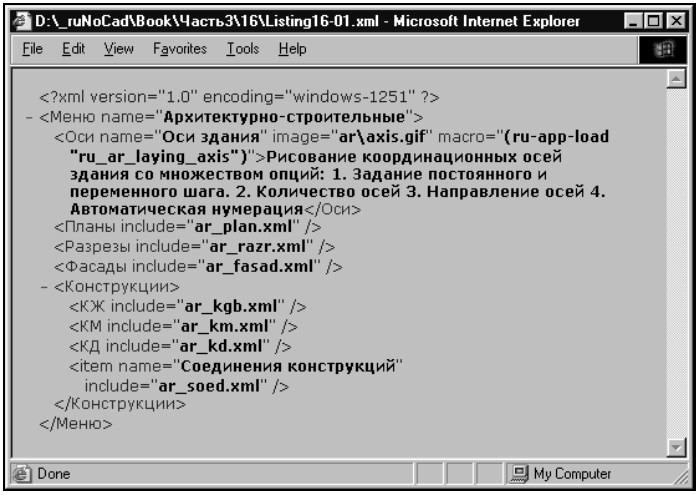


Рис. 16.1. Отображение XML-документа в браузере Internet Explorer

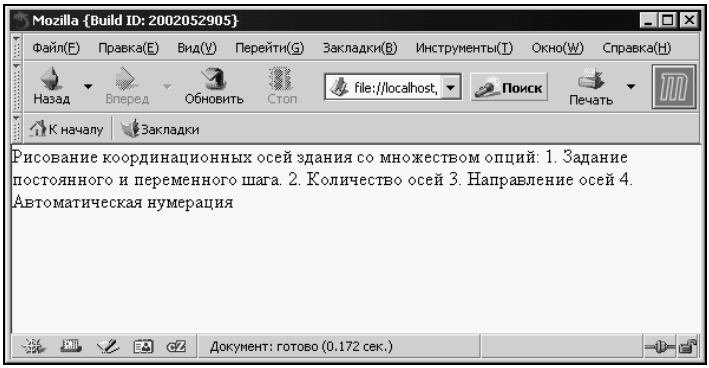


Рис. 16.2. Отображение XML-документа в браузере Mozilla

XML-документы обязательно должны быть корректно сформированы, но могут еще быть и *валидными*[2] (valid, по терминологии W3C). Валидный документ отвечает более

[1] В некоторых плохо переведенных изданиях применяется термин "правильный документ".

[2] В некоторых изданиях применяется термин "действительный документ".

строгому набору ограничений. Нам валидные документы не потребуются, и дополнительные требования мы рассматривать не будем.

Теперь запустим *нашу* программу `ruXmlTree` и загрузим в нее это меню. Мы увидим более подобающую картину.

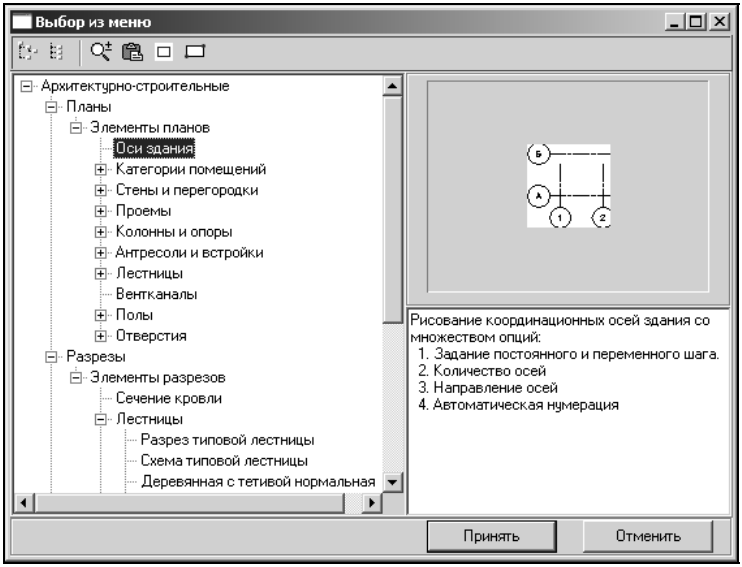


Рис. 16.3. Отображение XML-документа в программе `ruXmlTree`

Сравнивая текст XML-файла и окно программы, можно установить следующее:

- структура XML-файла отображается в дереве, а информация — в других компонентах окна;
- если элементы имеют атрибут `name`, в дереве отображается значение этого атрибута, иначе — имя элемента;
- в правой верхней панели у элементов, имеющих атрибут `image`, отображается иллюстрация;
- в правой нижней панели отображается пояснительный текст элементов;
- хотя в тексте документа имеются всего десять элементов, в дереве отображается несколько сотен узлов. Очевидно, что вложенные элементы подгружаются из других файлов, имена которых указаны в атрибутах `include`;
- при движении по дереву кнопка **Принять** иногда становится недоступной. "Следственный" эксперимент показал, что это происходит у элементов, не имеющих атрибута `macro`;
- оказывается, с помощью кнопок на панели инструментов можно заменить иллюстрацию, вставив ее из буфера или вырезав с экрана.

Экспериментальным путем мы выяснили правила работы программы. Осталось написать такую же. Но прежде разберемся, как же писать XML-документы.

Как создаются XML-документы

Ответ простой — руками. Лучше всего в обычном текстовом редакторе, желательно с подсветкой синтаксиса. Это относится к простым документам, которые мы будем создавать и использовать. Сложные документы создаются в специализированных редакторах, которые мы не будем рассматривать. Например, в редакторе XML Spy наш XML-документ будет выглядеть так, как показано на рис. 16.4.

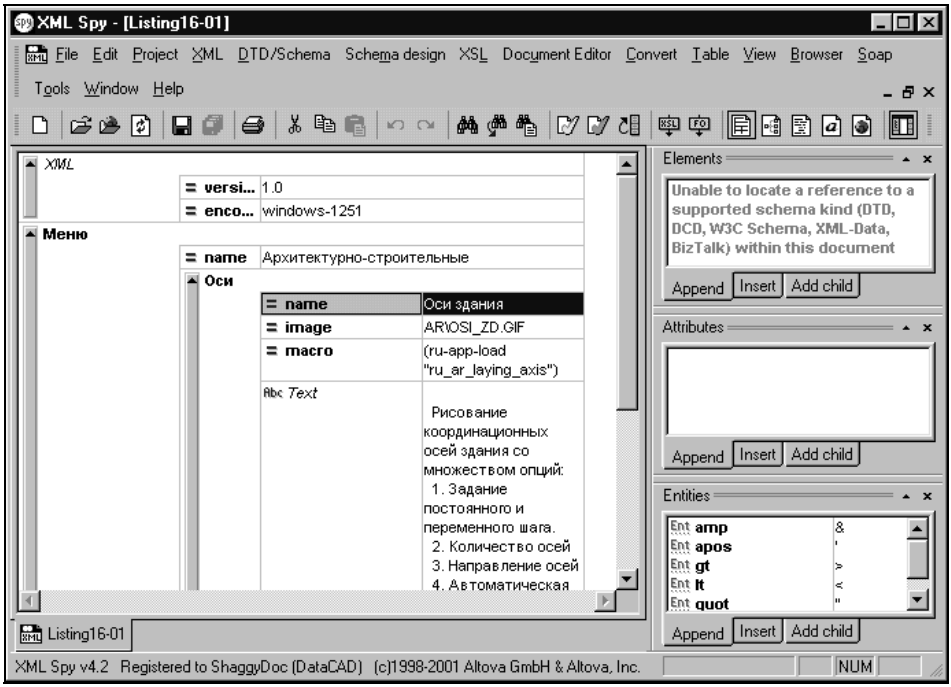


Рис. 16.4. Редактирование документа в редакторе XML Spy

Для своих документов мы создадим очень простой визуальный редактор, но только после того, как научимся писать XML "ручками". Создание XML-документов вручную позволит приобрести опыт работы, понять, как создаются элементы и атрибуты, позволит устранять неполадки, поскольку вы будете точно знать, что и где требуется. Опыт показывает, что даже пользователь категории "смышленая тетка" легко разбирается с текстом XML.

Огромное количество XML-документов создается автоматически при экспорте информации из баз данных. Этой темы мы еще коснемся. Система AutoCAD 2002 умела (а система AutoCAD 2004 делает вид, что уже разучилась) работать с блоками в формате DesignXML.

Проведем простой эксперимент в системе AutoCAD 2002.

Создадим в рисунке простой единичный блок (рис. 16.5) с именем RU_1 и запишем его на диск в формате DesignXML, для чего:

1. Вызовем команду WBLOCK (ПБЛОК).
2. В диалоговом окне **Write Block** (Запись блока на диск) (рис. 16.6) для имени файла *обязательно укажем расширение xml.*
3. Запишем блок на диск. В результате у нас появится файл ru_1.xml.

Откроем его в любой программе просмотра или текстовом редакторе (рис. 16.7). Текст понятный, но не очень "читабельный", но это только потому, что из него при автоматическом формировании были выброшены все ненужные пробелы.

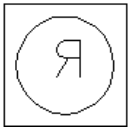


Рис. 16.5. Блок для экспорта в XML-файл

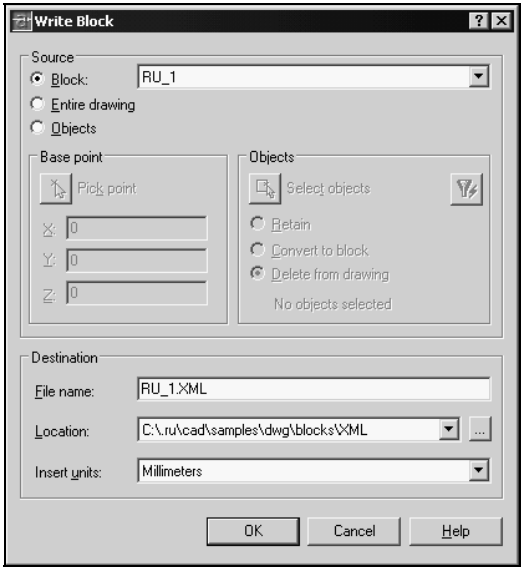


Рис. 16.6. Диалоговое окно экспорта блока в XML-файл

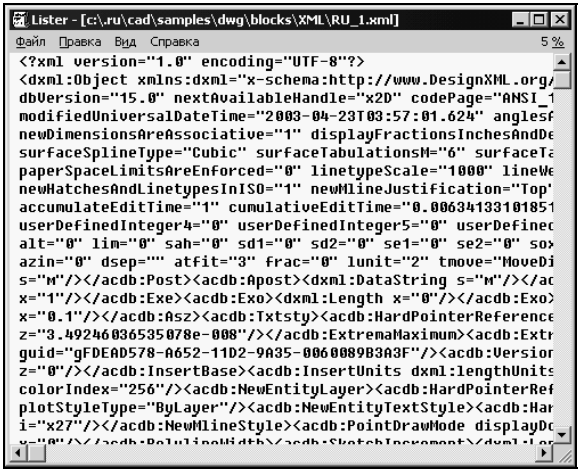


Рис. 16.7. Просмотр текста в формате XML с помощью программы Lister

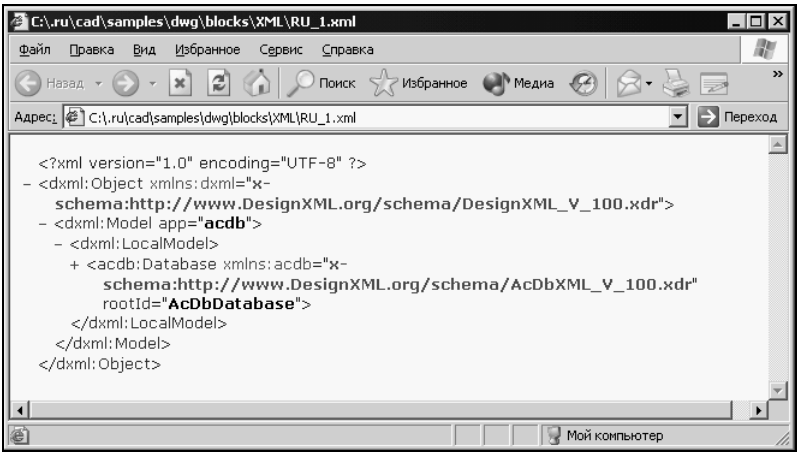


Рис. 16.8. Просмотр текста в формате DesignXML с помощью браузера Internet Explorer

При загрузке XML-файла в Internet Explorer мы видим совсем другую картину (рис. 16.8). Файл просматривается в виде дерева, ветви которого можно разворачивать и сворачивать. Фактически мы получили очень удобную навигацию по графической базе данных рисунка. Даже не зная формат DesignXML мы уже можем получить массу полезных сведений — все, что обычно записано в DWG-файл (даже картинку предпросмотра), но не в секретном двоичном формате, а в виде текста. Этот текст гораздо понятнее устаревшего формата обмена DXF и может быть прочитан любой "заинтересованной" программой. Internet Explorer "понятия не имеет" о существовании системы AutoCAD, структуре его файлов, да и о DesignXML тоже не знает.

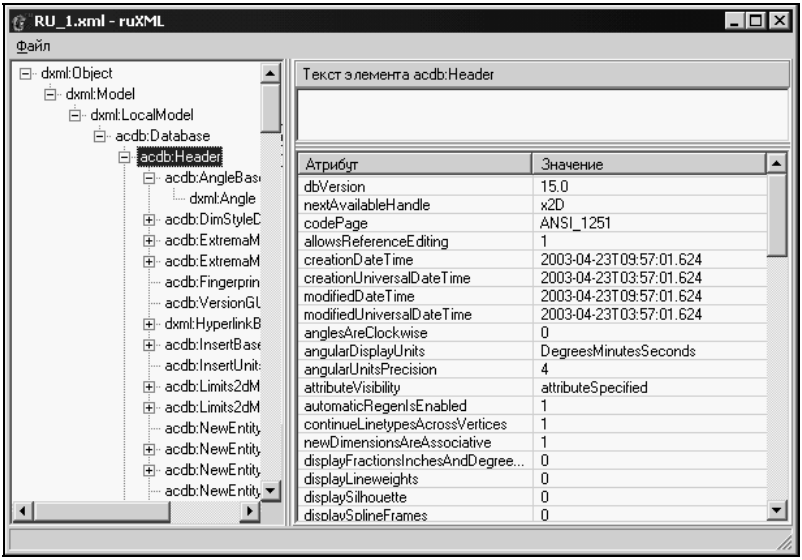


Рис. 16.9. Просмотр текста в формате DesignXML с помощью редактора ruXMLeditor

Он сумел отобразить этот документ потому, что он соответствует общим спецификациям XML и является корректным. Другие программы, которые могут отображать DesignXML в виде рисунка, очень скоро появятся. Пока это умеет делать только сама система AutoCAD.

Просмотр в нашей программе (которую мы когда-нибудь напишем) предоставит еще более удобную навигацию (рис. 16.9).

При просмотре этой захватывающей дух информации становится понятно, почему фирма Autodesk исключила возможности DesignXML из системы AutoCAD 2004 — похоже, что там испугались слишком удобного средства для изучения структуры графической базы данных. А может быть, это только наши домыслы, и на самом деле это сделано "для блага человека".

Как работают программы обработки XML

В мире существует множество редакторов и просмотрщиков XML-документов. Все они работают по одной схеме. При загрузке XML-документа производится синтаксический анализ (parsing) исходного текста. *Анализатор* (parser) проверяет соответствие кода документа спецификации XML. Анализатор обязан уметь проверять, является ли документ *корректно сформированным* (well-formatted), и может проверять, является ли документ *валидным* (valid). Анализаторы обычно представляют собой библиотеки, написанные на C++, Java, Delphi.

Если документ проверен, в действие вступает процессор обработки XML. Процессор формирует объектную модель документа и делает ее доступной для отображения различными визуальными компонентами. Часто анализатор кода и процессор совмещены. Такие библиотеки в дальнейшем мы будем называть *парсерами*.

Существует множество парсеров. Очень часто применяется парсер MSXML корпорации Microsoft. Популярность его объясняется добровольно-принудительной установкой в Windows и возможностью бесплатных обновлений. Это хороший парсер, умеющий делать все. Но его большие возможности являются и слабым местом. Часто эти возможности не нужны, а ресурсы на них потребляются. Кроме того, этот парсер, как и многие продукты Microsoft, является "слишком умным" и делает то, что от него совсем не требуется, а иногда просто является вредным. Например, в документе может быть изменена система кавычек — одинарные заменятся "без спроса" на двойные. Для наших программ это имеет большое значение из-за наличия кавычек в LISP-выражениях.

Перебрав множество продуктов, мы остановились на парсере janXmlParser2[1], умеющем делать именно то, что нам необходимо и не делать ничего ненужного.

Что должно делать наше приложение

Сформулируем требования к нашему приложению, с помощью которого мы будем осуществлять выбор из XML-меню. Эти требования сформировались на основании нескольких лет опыта эксплуатации, во время которого постоянно появлялись новые идеи.

---

[1] **www.jansfreeware.com**.

Итак, наше приложение должно уметь:

- отображать XML-документ в виде иерархического дерева;
- отображать для просмотра пользователем иллюстрацию и описание, которые может иметь любой элемент документа;
- учитывать права доступа пользователя (открывать документ в режиме "только для чтения" или предоставлять возможность пользователю, имеющему соответствующие права, изменять иллюстрацию, находящуюся во внешнем растровом файле);
- отображать вложенные файлы, которые могут быть в XML-документе;
- предоставлять возможность присвоения русских имен элементам и атрибутам; а также отображения в дереве не только коротких имен элементов, но и другого текста;
- предоставлять возможность программисту управлять всеми свойствами приложения;
- давать возможность выбора одного или нескольких элементов дерева;
- проверять возможность выбора элементов в зависимости от значения заданного программистом атрибута.

Реализация приложения для просмотра XML-меню

Имея технические требования нам осталось самое простое — реализовать их в работающем приложении. Выполнять задачу мы будем по следующей схеме:

1. Разработаем автономное приложение, добившись его правильной работы.
2. На базе автономного приложения создадим COM-сервер в виде DLL.
3. Из специального тестового приложения проверим и отладим работу COM-сервера.
4. Напишем функции для вызова приложения из Visual LISP.

Разработка автономного приложения

Разработку мы будем выполнять в Borland Delphi 6 в следующем порядке:

1. Создадим новый проект с одной пустой пока формой.
2. Сохраним проект под именем ruXMLmenu.dpr, а форму — под именем frmXmlMenu.pas.
3. Разместим на форме набор компонентов (рис. 16.10).

Происхождение и назначение некоторых компонентов, влияющих на логику работы приложения

Объект `formXMLmenuShow` (класс `TformXMLmenuShow`) — форма приложения. Используются события:

- `OnShow` = `FormShow` — устанавливаются свойства компонентов в зависимости от свойств формы и загружается XML-документ;
- `OnDestroy` = `FormDestroy` — освобождается объектная модель XML-документа.

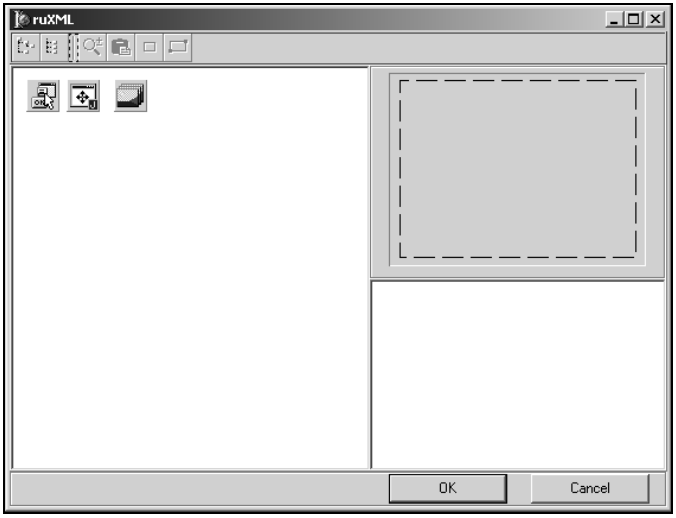


Рис. 16.10. Форма на этапе разработки

Объект `TreeXML` (класс `TJvTreeView`) — дерево для отображения документа. Использован компонент `JvTreeView` из бесплатной библиотеки JEDI VCL, которую мы будем применять очень часто, потому что этот компонент умеет, при необходимости, отображать "галочки" для пометки узлов в режиме множественного выбора. Используются события:

- `OnChange` = `TreeXMLChange` — основное событие при изменении выделенного узла дерева;
- `OnDblClick` = `TreeXMLDblClick` — введено для быстрого выбора выбранного элемента двойным щелчком, без нажатия на кнопку OK.

Объект `Memo1` (класс `TMemo`) — стандартный компонент для отображения текстов и комментариев.

Объект `ActionList1` (класс `TActionList`) — стандартный компонент для диспетчеризации действий программы. Компонент содержит список действий (объекты класса `TAction`, перечисленные далее). Каждое действие имеет свойства, такие как `Caption`, `Category`, `Enabled`, `Hint`, `ShortCut`, `ImageIndex`, и события (мы используем только событие `OnExecute`). Управляющие элементы типа кнопок, пунктов меню через свойство `Action`, имеющееся у этих элементов, могут быть связаны с действием. При этом у всех управляющих элементов устанавливаются такие же свойства, как и у соответствующего объекта класса `TAction`, а при щелчке по любому из этих управляющих элементов будет выполняться одно и то же событие `OnExecute`. Управляя доступностью объекта действия, мы одновременно управляем доступностью всех ссылающихся на него управляющих элементов. В нашем приложении в `ActionList1` входят следующие объекты действия:

- `aSelectProg: TAction`
  - `OnExecute = aSelectProgExecute` — событие, соответствующее нажатию кнопки `btnOk`;
- `aCloseXML: TAction`
  - `OnExecute = aCloseXMLExecute` — событие, соответствующее нажатию кнопки `btnCancel`;
- `aPasteFromClip: TAction`
  - `OnExecute = aPasteFromClipExecute` — событие, инициализирующее вставку иллюстрации из буфера обмена;
- `aPasteStandardImage: TAction`
  - `OnExecute = aPasteStandardImageExecute` — событие, инициализирующее копирование с экрана иллюстрации стандартного размера;
- `aPasteImageFree: TAction`
  - `OnExecute = aPasteImageFreeExecute` — событие, инициализирующее копирование с экрана иллюстрации произвольного размера;
- `aShowFullImage: TAction`
  - `OnExecute = aShowFullImageExecute` — событие, инициализирующее просмотр иллюстрации в отдельном окне в полный размер;
- `aExpandTree: TAction`
  - `OnExecute = aExpandTreeExecute` — событие, вызывающее раскрытие всех узлов дерева;
- `aCollapse: TAction`
  - `OnExecute = aCollapseExecute` — событие, вызывающее свертывание всех узлов дерева.

Объект *`Image`* (класс `TImage`) — стандартный компонент для отображения иллюстраций. Используется событие:

- `OnDblClick = aShowFullImageExecute` — введено для быстрого просмотра иллюстрации в отдельном окне в полный размер по двойному щелчку.

Объект *`btnOk`* (класс `TButton`) — стандартный компонент, принимающий выбор пользователя. Важнейшие свойства:

- `Action = aSelectProg`;
- `Default = True` — указывает, что именно эта кнопка является кнопкой по умолчанию, т. е. "нажимается" при нажатии клавиши <Enter>;
- `ModalResult = 1` — указывает, что в модальном диалоговом окне нажатие этой кнопки вернет результат `mrOk` (1);

Объект *`btnCancel`* (класс `TButton`) — стандартный компонент, закрывающий форму. Важнейшие свойства:

- `Action = aCloseXML`;
- `Cancel = True` — указывает, что эта кнопка работает аналогично клавише <Esc>;

- `ModalResult = 2` — указывает, что в модальном диалоговом окне нажатие этой кнопки вернет результат `mrCancel` (2).

Несколько управляющих кнопок на панели инструментов предназначены для ускорения действий пользователя. Каждая кнопка связана с одним из уже известных нам объектов действия:

- *объект* `ToolButtonExpand` (класс `TToolButton`), `Action` = `aExpandTree`;
- *объект* `ToolButtonCollapse` (класс `TToolButton`), `Action` = `aCollapse`;
- *объект* `ToolButtonShowFull` (класс `TToolButton`), `Action` = `aShowFullImage`;
- *объект* `ToolButtonPaste` (класс `TToolButton`), `Action` = `aPasteFromClip`;
- *объект* `ToolButtonPasteStd` (класс `TToolButton`), `Action` = `aPasteStandardImage`;
- *объект* `ToolButtonPasteFree` (класс `TToolButton`), `Action` = `aPasteImageFree`.

Исходный текст модуля

Исходный текст модуля приведен в листинге 16.2. Все, что мы вписали своими дрожащими ручками, выделено полужирным шрифтом. Остальной код сгенерирован средой Delphi автоматически. Исходный текст мы постарались достаточно подробно комментировать.

Листинг 16.2. Файл frmXmlTree.pas

```
unit frmXmlMenu;

interface
uses
{
Вручную вписываем модули, которые нам нужны, но о которых Delphi не знает.
Остальные вписывает сама Delphi.
Важнейшим из дополнительных модулей является janXMLparser2, который и выполняет
синтаксический анализ XML-документов.
AutoCAD_TLB мы внесли для того, чтобы, используя объектную модель, посылать
ценные указания прямо в командную строку AutoCAD.
}
ShellAPI, Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
janXMLparser2, ComCtrls, StdCtrls, ExtCtrls, ToolWin, Menus,IniFiles, janStrings,
Buttons, ActnList, JvPlacemnt, AutoCAD_TLB, JvCoolBar, JvSpeedButton, JvToolBar,
ImgList, JvComCtrls;

type
  TformXMLmenuShow = class(TForm)
    PanelButton: TPanel;
    ActionList1: TActionList;
    aSelectProg: TAction;
    aCloseXML: TAction;
    aPasteFromClip: TAction;
    aPasteSelection: TAction;
    aPasteStandardImage: TAction;
    aPasteImageFree: TAction;
```

```
    aShowFullImage: TAction;
    aExpandTree: TAction;
    aCollapse: TAction;
    btnOk: TButton;
    btnCancel: TButton;
    JvFormPlacement1: TJvFormPlacement;
    PanelAll: TPanel;
    PanelData: TPanel;
    PanelImage: TPanel;
    BevelImage: TBevel;
    Image: TImage;
    Memo1: TMemo;
    JvToolBar1: TJvToolBar;
    ToolButtonExpand: TToolButton;
    ToolButtonCollapse: TToolButton;
    ToolButton3: TToolButton;
    ToolButtonShowFull: TToolButton;
    ToolButtonPaste: TToolButton;
    ToolButtonPasteStd: TToolButton;
    ToolButtonPasteFree: TToolButton;
    ImageList1: TImageList;
    TreeXML: TJvTreeView;
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure FormShow(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure TreeXML1Change(Sender: TObject; Node: TTreeNode);
    procedure aShowFullImageExecute(Sender: TObject);
    procedure aCollapseExecute(Sender: TObject);
    procedure aExpandTreeExecute(Sender: TObject);
    procedure aSelectProgExecute(Sender: TObject);
    procedure aCloseXMLExecute(Sender: TObject);
    procedure aPasteFromClipExecute(Sender: TObject);
    procedure aPasteStandardImageExecute(Sender: TObject);
    procedure aPasteImageFreeExecute(Sender: TObject);
    procedure TreeXMLDblClick(Sender: TObject);
  private
    { Private declarations }
{
Здесь мы вписали дополнительные процедуры и функции, которые добавляем сами
 }
    procedure OpenFile;
    procedure ParseTree;
    procedure ParseTreeNode(TreeNode: TTreeNode; DomNode: TjanXMLNode2);
    function GetAttribValue(node: TjanXMLNode2; AttribName,
      Default:string):string;
  public
    { Public declarations }
    ImageFile,ImageNote: string;
```

```
{
Эта группа переменных будет устанавливаться через свойства COM-объекта
}
{
 Имя загружаемого документа
}
  XMLFileName: string;
{ Корневой каталог, относительно которого указываются адреса вложенных документов}
  XMLRootDir: string;
{ Каталог, в котором находятся иллюстрации к пунктам меню}
  XMLImagesDir: string;
{
Имя INI-файла, в который будут записываться результаты выбора. Возврат результатов
мы предусматриваем именно через файл, несмотря на то, что такой способ презираем
"настоящими" программистами, работающими исключительно через память.
Причина в том, что вернуть множество атрибутов и их значений через память
затруднительно, особенно из-за того, что Visual LISP не поддерживает прямое чтение
списков строк. Кроме того, это приложение будет применяться нами не только в
AutoCAD, но и в других системах, где обмен данными через память вообще невозможен.
Для простого случая, когда необходимо вернуть в AutoCAD строку макрокоманды, у нас
предусмотрен специальный режим.
}
  ResIniFileName: string;
{ Свойство, позволяющее разрешить доверенным пользователям "на лету" заменять
иллюстрации }
  CaptureImageEnabled: boolean;
{ Свойство, разрешающее выбор нескольких узлов в зависимости от логики основного
приложения }
  MultiSelect: boolean;
{ Имя атрибута, значение которого отображается в дереве вместо имени элемента.
Позволяет представлять структуру в более читабельном виде }
  TreeViewedAttrib: string;
{ Имя атрибута, значением которого является относительное имя включаемого
XML-документа }
  IncludeAttrib: string;
{ Имя атрибута, значением которого является относительное имя растрового файла,
отображаемого в виде иллюстрации }
  ImageAttrib: string;
{ Имя атрибута, значение которого отображается в Memo1 при отсутствии текста
элемента }
  CommentAttrib: string;
{ Имя атрибута, наличие которого делает доступной кнопку Ok }
  RequiredAttrib: string;
{ Заголовок окна приложения }
  DlgCaption: string;
{ Свойство, позволяющее отправить в AutoCAD значение атрибута, имя которого задано
в свойстве RequiredAttrib. При установке SendAcadEnabled в True приложение
соединяется с AutoCAD и посылает в его командную строку значение атрибута.
Обычно это LISP-выражение или команда AutoCAD.
Вообще-то это может быть совсем не AutoCAD, а другое приложение, имя которого
задается значением атрибута AcadAppString. Естественно, для другого приложения
должны быть и свои макросы }
  SendAcadEnabled: boolean;
```

```
{ Строка, указывающая, с каким сервером автоматизации необходимо связываться. Может
быть дополнительно указана версия AutoCAD, например
'AutoCAD.Application.16'
}
  AcadAppString: string;
{ Номер узла, выбиравшегося в прошлый раз. Позволяет открыть дерево на прежнем
месте}
  LastSelect: integer;
{ Имя INI-файла для сохранения размеров и местоположения формы, восстанавливаемых
в следующем сеансе }
  StorageIniFile: string;
{ Заголовок кнопки Ok по смыслу программы}
  OkButtonCaption: string;
{ Заголовок кнопки Cancel по смыслу программы }
  CancelButtonCaption: string;
 end;

var
  formXMLmenuShow: TformXMLmenuShow;
{
Важнейшая переменная - объектная модель XML-документа
}
  Dom: TjanXMLParser2;
{
Наш старый знакомый - AutoCAD. Надо же и с ним пообщаться...
}
  AcadApplication: IAcadApplication;

implementation

uses ComObj, Clipbrd, GifImage, CaptureForm, TMBmpView, JPEG;

{$R *.DFM}

////////////////////////////////////////////////////////
procedure TformXMLmenuShow.FormShow(Sender: TObject);
{
Эта процедура является основной в работе приложения
}
begin
{
Устанавливаем видимость действий и связанных с ними управляющих элементов
в зависимости от разрешения на изменение иллюстраций
}
  aPasteFromClip.Visible := CaptureImageEnabled;
  aPasteSelection.Visible := CaptureImageEnabled;
  aPasteStandardImage.Visible := CaptureImageEnabled;
  aPasteImageFree.Visible := CaptureImageEnabled;
{//////////////////////////////////////////
Для варианта прямого соединения с AutoCAD. Обратите внимание - на форме у нас нет
никаких компонентов AutoCAD! Мы создаем объект AcadApplication динамически.
```

```
Но т. к. мы сослались на модуль AutoCAD_TLB.pas, Delphi знает свойства и методы
AcadApplication, показывает их в Code Insight и компилятор не пропустит неизвестных
свойств и методов
////////////////////////////////////////////////}
  if SendAcadEnabled then begin
    Try
     {
     Пытаемся найти активный AutoCAD
     }
      AcadApplication := GetActiveOleObject(AcadAppString)
                         as IAcadApplication;
    except
     {
     Если работающий AutoCAD не найден
     }
      try
      {
       Пытаемся запустить AutoCAD
      }
        AcadApplication := CreateOleObject(AcadAppString)
                           as IAcadApplication;
      except
        MessageDlg('AutoCAD вообще не найден', mtError, [mbOK], 0);
        Halt;
      end;
    end;
    if AcadApplication = nil then
    begin
      btnOk.Enabled := False;
      MessageDlg('Ошибка соединения с AutoCAD', mtError, [mbOK], 0);
    end
    else
      btnOk.Enabled := True;
  end;
{
Устанавливаем свойства других компонентов
}

  btnOk.Caption := OkButtonCaption;
  btnCancel.Caption := CancelButtonCaption;
  Caption := DlgCaption;
  JvFormPlacement1.IniFileName := StorageIniFile;
  JvFormPlacement1.IniSection := XMLFileName;
  {Восстанавливаем прежние размеры и положение формы}
  JvFormPlacement1.RestoreFormPlacement;
  ImageFile := '';
  ImageNote := '';
  TreeXML.Checkboxes := MultiSelect;
  TreeXML.MultiSelect := MultiSelect;
```

```
{//////////////////////////////////
Важнейшая операция – создаем объект модели XML-документа
//////////////////////////////////}
  dom := TjanXMLparser2.Create;
  if fileexists(XMLFileName) then
{
Открываем документ
}
     OpenFile
    else
     Application.MessageBox(pchar('Не найден файл ' + XMLFileName),
                                'Ошибка!', MB_OK or MB_ICONERROR);
end;

////////////////////////////////////////////////////////
procedure TformXMLmenuShow.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  JvFormPlacement1.SaveFormPlacement;
end;
////////////////////////////////////////////////////////
procedure TformXMLmenuShow.OpenFile;
{//////////////////////////////
Процедура открытия XML-документа.
//////////////////////////////}
begin
{Загружаем документ в парсер}
  dom.LoadXML(XMLFileName);
{Готовим компонент дерева}
  TreeXML.items.BeginUpdate;
  TreeXML.items.clear;
{//////////////////////////////
Выполняем анализ дерева документа и связываем его с визуальными компонентами
//////////////////////////////}
  ParseTree;
  TreeXML.items.EndUpdate;
  TreeXML.FullCollapse;
{
Раскрываем дерево и устанавливаем фокус на узел, выбиравшийся в предыдущем сеансе
}
  if TreeXML.Items.Count > LastSelect then
    if LastSelect > -1 then
    begin
      TreeXML.FullCollapse;
      TreeXML.Items.Item[LastSelect].MakeVisible;
      TreeXML.Items.Item[LastSelect].Selected := true;
    end;
end;
```

```
////////////////////////////////////////////////////////////
function TformXMLmenuShow.GetAttribValue(node: TjanXMLNode2; AttribName,
                                         Default:string): string;
{////////////////////////////////////////////
Функция получения значения атрибута узла модели документа по имени атрибута
////////////////////////////////////////////}
var
  attrib: TjanXMLattribute2;
  i, attr_count: integer;
begin
  result := Default;
  if node.hasAttribute(AttribName) then
  begin
    attr_count := node.attributes.Count;
    for i := 0 to attr_count - 1 do
    begin
      attrib := TjanXMLAttribute2(node.attributes[i]);
      if attrib.name = AttribName then
        result := attrib.value;
    end;
  end;
end;

////////////////////////////////////////////////////////////
procedure TformXMLmenuShow.FormDestroy(Sender: TObject);
// Уничтожение формы с освобождением модели документа
begin
  dom.free;
end;

procedure TformXMLmenuShow.TreeXMLChange(Sender: TObject; Node: TTreeNode);
{
Эта процедура вызывается при перемещении фокуса на другой узел дерева
}
var
  { Это объект узла документа - не путать с узлом дерева}
  DomNode: TjanXMLNode2;
  RequiredAttribValue,
  Comment: string;
begin
{
Сбрасываем прежние значения и блокируем все действия
}
  ImageFile := '';
  ImageNote := '';
  aShowFullImage.Enabled := False;
  aSelectProg.Enabled := False;
```

```
  aPasteFromClip.Enabled := False;
  aPasteSelection.Enabled := False;
  aPasteStandardImage.Enabled := False;
  aPasteImageFree.Enabled := False;
{
Создаем указатель на узел модели документа, связанный с текущим узлом дерева
}
  DomNode := TjanXMLNode2(Node.data);
  if TreeXML.selected <> nil then
  begin
{ Удаляем прежнюю иллюстрацию }
    image.Picture := nil;
{ Выясняем имя файла иллюстрации текущего узла }
    ImageFile := GetAttribValue(DomNode, ImageAttrib, '');
    RequiredAttribValue := GetAttribValue(DomNode, RequiredAttrib, '');
{
Проверяем наличие непустого значения требуемого атрибута и, если такое найдено,
разблокируем кнопку OK
}
    if RequiredAttribValue <> '' then
      aSelectProg.Enabled := True;
    if ImageFile <> '' then begin
      ImageFile := XMLImagesDir + ImageFile;
      if FileExists(ImageFile) then
      begin
{
Если с узлом связан файл иллюстрации и если этот файл существует, разблокируем все
действия, связанные с иллюстрацией. Действия, связанные с изменением иллюстрации,
могут остаться невидимыми.
Загружаем иллюстрацию
}

        image.picture.LoadFromFile(ImageFile);
        ImageNote := GetAttribValue(DomNode, TreeViewedAttrib,
                        DomNode.name);
        aShowFullImage.Enabled := True;
        aPasteFromClip.Enabled := CaptureImageEnabled;
        aPasteSelection.Enabled := CaptureImageEnabled;
        aPasteStandardImage.Enabled := CaptureImageEnabled;
        aPasteImageFree.Enabled := CaptureImageEnabled;
      end;
    end;
{
Далее разбираемся с комментариями, отображаемыми в Memo.
Если у элемента XML имеется текст, то в Memo отображается текст, иначе проверяется
наличие атрибута, в котором могут быть комментарии, и, если комментарий найден,
отображается эта строка
}
```

```
    Memo1.Lines.Clear;
    Comment := GetAttribValue(DomNode, CommentAttrib, '');
    if DomNode.text <> '' then
      memo1.Lines.Add(DomNode.text)
    else if Comment <> '' then
      memo1.Lines.Add(Comment);
  end;
end;

//////////////////////////////////////////////////////////
procedure TformXMLmenuShow.ParseTree;
{
Разборка модели документа и связывание ее узлов с визуальным деревом
}
var
  treeNode: TTreeNode;
  DOMnode: TjanXMLNode2;
  nodename: string;
begin
{
Первый узел берет указатель на всю модель документа, т. е. на ее корень
}
  DOMnode := dom;
{
Получаем имя элемента
}
  nodename := DOMnode.name;
{
Проверяем, нет ли у узла атрибута, значение которого должно отображаться в дереве
вместо имени элемента. Если такого атрибута нет, или он пустой, то именем узла
останется имя элемента
}
  nodename := GetAttribValue(DOMnode, TreeViewedAttrib, nodename);
{
Создаем первый корневой узел визуального дерева. Добавляем объект узла дерева,
родителем которого является nil (это будет корневой узел). Этому узлу присваивается
имя корневого элемента модели документа и с ним связывается указатель на корневой
узел модели документа
}
  treeNode := TreeXML.Items.AddObject(nil, nodename, DOMnode);
{
Далее вновь созданный узел визуального дерева и корневой узел модели передаются
в процедуру разборки узла.
}
  ParseTreeNode(treeNode, DOMnode);
end;

//////////////////////////////////////////////////////////
procedure TformXMLmenuShow.ParseTreeNode(TreeNode: TTreeNode;
                                         DomNode: TjanXMLNode2);
```

```
{//////////////////////////////////////////////////
Важнейшая процедура разборки узла. В качестве параметров передаются указатели
на текущий узел визуального дерева и текущий узел модели документа
//////////////////////////////////////////////////}
var
  DomInclude: TjanXMLParser2;
  TreeNodeNew, TreeNodeInclude: TTreenode;
  DomNodeInclude, DomNodeNew: TjanXMLNode2;
  i, DomNodeNodesCount: integer;
  IncludeXML: string;
Begin
{
Выясняем количество узлов модели документа, вложенных в ее разбираемый узел
}
  DomNodeNodesCount := DomNode.Nodes.Count;
  if DomNodeNodesCount = 0 then
    exit;
{
Разбираем каждый вложенный узел модели документа
}
  for i := 0 to DomNodeNodesCount - 1 do
  begin
{
Создаем для каждого узла модели документа новый указатель
}
    DomNodeNew := TjanXMLNode2(DomNode.Nodes[i]);
{
Добавляем в визуальное дерево к текущему узлу потомка с именем, извлекаемым
из значения атрибута
}
    TreeNodeNew := TreeXML.items.AddChild(TreeNode,
    GetAttribValue(DomNodeNew, TreeViewedAttrib,DomNodeNew.name));
{
Привязываем к узлу дерева указатель на соответствующий узел модели документа
}
    TreeNodeNew.data := DomNodeNew;
{
Выясняем, нет ли в узле модели документа ссылки на вложенный XML-файл
}
    IncludeXML := GetAttribValue(DomNodeNew, IncludeAttrib, '');
    if IncludeXML <> '' then
    begin
{
Если ссылка на вложенный файл есть, проверяем его наличие. Но сначала нужно
вычислить полное имя включаемого файла. Это очень важно, т. к. при относительном
имени (а именно оно записано в XML) включаемые файлы будут разыскиваться в каталоге
основной программы.
}
```

```
      IncludeXML := IncludeTrailingBackslash(XMLRootDir)+IncludeXML;
      if FileExists(IncludeXml) then
      begin
{
Если вложенный файл найден, создаем новую модель документа для включаемого файла и
загружаем включаемый файл
}
        DomInclude := TjanXMLParser2.create;
        DomInclude.LoadXML(IncludeXML);
{
Создаем для включаемой модели документа узел в текущей модели
}
        DomNodeInclude := TjanXMLNode2.create;
{
Устанавливаем узел текущей модели родителем для вновь созданной вложенной модели
документа
}
        DomNodeInclude.ParentNode := DomNodeNew;
{
Параллельно добавляем потомка и к узлу визуального дерева в качестве потомка
}
        TreeNodeInclude := TreeXML.Items.AddChildObject(TreeNodeNew,
          GetAttribValue(DomInclude,
          TreeViewedAttrib, DomInclude.name), DomNodeNew);
{
И производим рекурсивную разборку включенной модели документа, начиная с корневого
узла
}
        ParseTreeNode(TreeNodeInclude, DomInclude);
      end;
    end;
{
В обычном узле модели документа также производится рекурсивная разборка.
}

    ParseTreeNode(TreeNodeNew, DomNodeNew);
  end;
end;

//////////////////////////////////////////////////////////
procedure TformXMLmenuShow.aShowFullImageExecute(Sender: TObject);
{Просмотр рисунка в полный размер}
begin
  if ImageFile <> '' then
{
процедура ShowImageWin находится в отдельном модуле. В книге ее не приводим, т. к.
там нет ничего интересного
}
    ShowImageWin(ImageFile, ImageNote);
end;
```

```
////////////////////////////////////////////////////////
procedure TformXMLmenuShow.aCollapseExecute(Sender: TObject);
begin
  TreeXML.FullCollapse;
end;

////////////////////////////////////////////////////////
procedure TformXMLmenuShow.aExpandTreeExecute(Sender: TObject);
begin
  TreeXML.FullExpand;
end;

////////////////////////////////////////////////////////
procedure TformXMLmenuShow.aSelectProgExecute(Sender: TObject);
{//////////////////////////////////////////////////
Важная процедура обработки кнопки OK
//////////////////////////////////////////////////}
var
  macro: string;
  DomNode: TjanXMLNode2;
  i, k, selcount: integer;
  ini: TIniFile;
  node: TTreeNode;
  LastSelected: integer;
begin
 if SendAcadEnabled then
{
Это вариант непосредственной отправки результата выбора в командную строку AutoCAD
}
  begin
{
Получаем указатель на узел модели документа, связанный с выделенным узлом
визуального дерева
}
    DomNode := TjanXMLNode2(TreeXML.Selected.data);
{
Извлекаем строку, являющуюся значением требуемого атрибута. По логике работы это
LISP-выражение или команда AutoCAD, т. е. то, что ДОЛЖНО правильно быть понятым
системой AutoCAD
}
    macro := GetAttribValue(DomNode, RequiredAttrib, '');
{
На всякий случай активируем текущий документ в AutoCAD
}
    AcadApplication.ActiveDocument.Activate;
{
Посылаем макрос в командную строку, добавляя нажатие <Enter>
}
    AcadApplication.ActiveDocument.SendCommand(macro + #13);
```

```
{
Больше ничего не делаем, реакцию AutoCAD на полученный "привет" не анализируем.
}
  end else
  begin
{
А это вариант обычной работы. Результаты записываются в INI-файл, а их анализ
возлагается на основное приложение
}
    lastselected := TreeXML.Selected.AbsoluteIndex;
{
Первым делом записываем номер выбранного узла для последующего открытия дерева
в этом же месте
}
    ini := tinifile.Create(ExpandFileName(ResIniFileName));
    ini.writeInteger('result', 'lastselected', lastselected);
    ini.Free;
{
Для варианта множественного отбора
}
    if multiselect then
    begin
      if TreeXML.SelectedCount > 0 then
      begin
        ini := tinifile.Create(ResIniFileName);
        selcount := 0;
        for k := 0 to TreeXML.Items.Count - 1 do
          if TreeXML.GetChecked(TreeXml.Items[k]) then
{
Для каждого узла визуального дерева, помеченного "галочкой"
}
          begin
            selcount := selcount + 1;
            node := TreeXML.Items[k];
            ini.EraseSection('item' + inttostr(k + 1));
            DomNode := TjanXMLNode2(node.data);
{
Для каждого атрибута, имеющегося у узла
}
            for i := 1 to DomNode.attributecount do
              if DomNode.attribute[i - 1] <> '' then
{
Записываем в специальную секцию
}
                ini.writestring('item' + inttostr(selcount),
{
Имя переменной - имя атрибута
}
                  DomNode.attributename[i - 1],
```

```
{
Значение переменной - значение атрибута
}
                  DomNode.attribute[i - 1])
              else
{
А это на случай, если у атрибута нет значения - пишем пустое значение
}
                ini.writestring('item' + inttostr(selcount),
                  DomNode.attributename[i - 1], '');
          end;
{
В конце записываем количество выделенных узлов
}
        ini.writeinteger('multiselect', 'SelectionCount', selcount);
        ini.Free;
      end;
    end
    else
    begin
{
Вариант с одиночным выбором
}
      ini := tinifile.Create(ExpandFileName(ResIniFileName));
      DomNode := TjanXMLNode2(TreeXML.Selected.data);
      for i := 1 to DomNode.attributecount do
        if DomNode.attribute[i - 1] <> '' then
{
Записываем в секцию в качестве переменных имена всех атрибутов и их значений
}
          ini.writestring('singleselect',
            DomNode.attributename[i - 1],
            DomNode.attribute[i - 1])
        else
          ini.writestring('singleselect',
            DomNode.attributename[i - 1],
            '');
      ini.Free;
    end;
   end;
{
Форму закрываем во всех случаях, в том числе при отправке в AutoCAD. Если при
отправке команды в AutoCAD форму не закрывать, то у него в командной строке
создастся очередь, ведь форма из него вызвана. Для EXE-сервера, работающего
в отдельном пространстве имен, мог бы быть другой вариант.
}
 Close;
 ModalResult := mrOk;
end;
```

```
//////////////////////////////////////////////////////////
procedure TformXMLmenuShow.aCloseXMLExecute(Sender: TObject);
begin
  Close;
  ModalResult := mrCancel;
end;
//////////////////////////////////////////////////////////
procedure TformXMLmenuShow.aPasteFromClipExecute(Sender: TObject);
{
Вставка иллюстрации из буфера обмена
}

var
  ext: string;
  j: TJPEGImage;
  gif: TGIFImage;

begin
  if clipboard = nil then
  begin
    Application.MessageBox(pchar('Буффер пуст!'), 'Ошибка!', MB_OK or
      MB_ICONERROR);
    exit;
  end;
  if not clipboard.HasFormat(cf_bitmap) then
  begin
    Application.MessageBox(pchar('Неверный формат буфера!'), 'Ошибка!', MB_OK or
      MB_ICONERROR);
    exit;
  end;
  try
    image.Picture := nil;
    image.Picture.Assign(clipboard);
{
Связываем изображение объекта Image с буфером обмена, если буфер не пуст и в нем
изображение
}
    if extractfilename(ImageFile) = '' then
    begin
      MessageDlg('Файл не может быть сохранен!',
        mtConfirmation, [mbYes], 0);
      exit;
    end;
{
Сохраняем изображение в файле с прежним именем и расширением
}
    ext := ansilowercase(extractfileext(ImageFile));
    if ext = '.bmp' then
```

```
    begin
      image.Picture.SaveToFile(ImageFile);
      exit;
    end;
    if ext = '.gif' then
    begin
      GIF := TGIFImage.Create;
      try
        GIF.Assign(image.Picture);
        GIF.SaveToFile(ImageFile);
      finally
        GIF.Free;
      end;
    end
    else
    begin
      if (ext = '.jpg') or (ext = '.jpeg') then
      begin
        j := TJPEGImage.Create;
        try
          j.Assign(image.Picture.bitmap);
          j.CompressionQuality := 75;
          j.Compress;
          j.SaveToFile(ImageFile);
        finally
          j.Free;
        end;
      end
      else
      begin
        image.Picture.SaveToFile(ImageFile);
      end;
    end;
{
Вновь загружаем изображение из обновленного файла
}
    image.Picture.LoadFromFile(ImageFile);
  finally
  end;
end;

//////////////////////////////////////////////////////////
procedure TformXMLmenuShow.aPasteStandardImageExecute(Sender: TObject);
{
Процедура позволяет вырезать с экрана иллюстрацию "стандартного" размера, т. е.
такого, как у объекта Image. В прежних версиях стандартной была картинка размером
64x64 пиксела, такие картинки иногда показаны на иллюстрациях к книге.
}
```

```
var
  ind: integer;
begin
  ind := TreeXml.Selected.AbsoluteIndex;
{
Наше приложение скрывается
}
  Hide;
  Application.ProcessMessages;
  Sleep(500);
{
Выполняется CaptureScreenToFile, описанная в отдельном модуле. На экране появляется красная рамка заданного размера. Рамкой выбирается нужный участок экрана (обычно из графической области AutoCAD) и нажимается Enter. Вырезанное изображение сохраняется в файл
}
  if CaptureScreenToFile(Image.Width, Image.Height, ImageFile) then
    {UpdateNodeImages(TreeView.Selected)}
   ;
{
Восстанавливается форма
}
  show;
{
Восстанавливается выделенный узел дерева
}
  TreeXml.Selected := TreeXml.Items.Item[ind];
{
Изображение в Image загружается из обновленного файла
}
  image.Picture.LoadFromFile(ImageFile);
end;

//////////////////////////////////////////////////////////
procedure TformXMLmenuShow.aPasteImageFreeExecute(Sender: TObject);
{
Процедура аналогична предыдущей, только можно вырезать рамку произвольного размера
}
var
  ind: integer;
begin
  ind := TreeXml.Selected.AbsoluteIndex;
  Hide;
  Application.ProcessMessages;
  Sleep(500);
{
В этот момент нужно указать точку и растягивать рамку до нужного размера
}
```

```
  if CaptureScreenToFileFreeSize(ImageFile) then
   ;
  Show;
  TreeXml.Selected := TreeXml.Items.Item[ind];
  image.Picture.LoadFromFile(ImageFile);
end;

//////////////////////////////////////////////////////////
procedure TformXMLmenuShow.TreeXMLDblClick(Sender: TObject);
{
Процедура, позволяющая произвести одиночный выбор двойным щелчком по узлу
визуального дерева
}
begin
  if aSelectProg.Enabled then aSelectProgExecute(Sender);
end;
end.
```

Разработка автономного приложения закончена. Будем считать, что мы его протестировали в работе. Такую форму можно использовать и в самостоятельной программе. Дотошные читатели, конечно, заметили, что нигде нет инициализации переменных. Инициализацию надо бы вставлять в процедуру `TformXMLmenuShow.FormCreate`, которую мы просто не привели.

Разработка COM-сервера в виде DLL

Автономное приложение нам не особенно нужно. Нам требуется, в конце концов, иметь функцию в Visual LISP для работы с таким объектом. Следовательно, нам необходим COM-сервер, к которому можно обращаться из Visual LISP точно так же, как мы обращаемся к объектной модели AutoCAD. Давайте займемся его созданием.

Шаг 1. В среде Borland Delphi мы выбираем пункт меню **File | New** (Файл | Новый).

Шаг 2. В появившемся диалоговом окне на вкладке **ActiveX** выбираем **ActiveX Library** (Библиотека ActiveX).

Шаг 3. Сохраняем созданный проект под именем ruXmlMenuSrv.dpr в том же каталоге, где находится frmXmlMenu.pas.

Шаг 4. Выбираем пункт меню **File | New** (Файл | Новый). На вкладке **ActiveX** выбираем **Automation Object** (Объект автоматизации).

Шаг 5. В открывшемся диалоговом окне **Automation Object** (Объект автоматизации) в поле **CoClassName** (Имя класса) вводим `XmlTree` и щелкаем по кнопке **OK**. Открывается окно редактора библиотеки типов с корневым элементом `ruXmlMenuSrv`, имеющим в наследниках `interface IxmlTree CoClass XmlTree`.

Шаг 6. Переходим к наполнению объекта автоматизации свойствами и методами.

Здесь возможны варианты. Можно, например, предусмотреть свойства `SendAcadEnabled`, `CaptureImageEnabled`, `MultiSelect` и т. п. для инициализации переменных, объявленных в секции `public` формы. Именно эти свойства должны быть доступны приложениям-клиентам. Затем следует предусмотреть методы для создания объекта, вызова

диалога и для удаления объекта. Программист в прикладном приложении должен был бы последовательно создать объект, установить свойства, вывести диалоговое окно, удалить объект. Изменение свойств во время работы для подобных приложений не имеет смысла (такого, как, например, изменение цвета экрана у системы AutoCAD).

Для добавления свойств необходимо:

1. Выделить в дереве `IxmlTree`.
2. В панели инструментов у групповой кнопки **New Property** (Новое свойство) выбрать **WriteOnly** (Только запись).
3. У появившегося в дереве свойства изменить имя по умолчанию на очередное из приведенного выше списка (прямо в дереве или на вкладке **Attributes** (Атрибуты)).
4. На вкладке **Attributes** (Атрибуты) установить тип свойства (для типа string — BSTR, для типа boolean — VARIANT_BOOL (рис. 16.11).

При добавлении свойств можно наблюдать, что на вкладке **Parameters** (Параметры) каждое свойство получит параметр `Param1` с типом `BSTR` или `VARIANT_BOOL` и `Modifier` [`in`]. Менять эти значения не нужно, так же, как и значение раскрывающегося списка **Return Type** (Тип возврата) `HRESULT`.

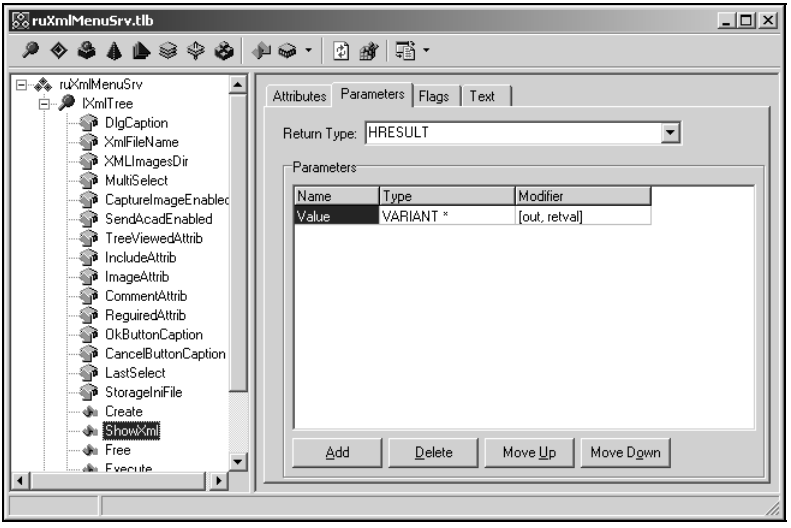


Рис. 16.11. Редактирование библиотеки типов

Шаг 7. После добавления свойств необходимо добавить методы. Добавление методов производится из панели инструментов или из контекстного меню выбором **New | Method** (Новый | Метод). После добавления имя метода по умолчанию заменяется на требуемое. В нашем приложении мы создадим такие методы:

- `Create` — создание объекта автоматизации (параметров нет, **Return Type** (Тип возврата) `HRESULT`);

- `ShowXml` — вывод на экран формы (параметр `Value` типа `VARIANT*`, `Modifier [out, retval]`, **Return Type** (Тип возврата) `HRESULT`);
- `Free` — уничтожение формы (параметров нет, **Return Type** (Тип возврата) `HRESULT`);
- `Execute` — выполнение диалога за "один присест" (параметров 16 штук, соответствующих по именам и типам ранее введенным свойствам, **Return Type** (Тип возврата) `HRESULT`).

Первые три метода предполагают, что из приложения-клиента создается объект автоматизации методом `Create`, затем устанавливаются все или некоторые свойства, далее вызывается метод `ShowXml`, клиент получает результат, при необходимости изменяются какие-либо свойства, снова вызывается метод `ShowXml` и в конце сеанса вызывается метод `Free`. При такой технологии возможно из LISP открыть диалоговое окно и, не закрывая его, периодически посылать в систему AutoCAD команды. Закрытие диалогового окна выполняется отдельной командой.

> А теперь все, что сделано в шаге 6, ликвидируем! Добавление свойств мы произвели в учебном режиме. Удаляем все свойства и методы, кроме `Execute`!

Метод `Execute` является классической функцией, которой нужно передать все параметры диалогового окна, которое сразу показывается и, после выбора параметров, закрывается. При его использовании разработчик приложения-клиента просто не имеет возможности допустить такие ошибки, как вывод диалогового окна до инициализации свойств или уничтожение не созданного объекта.

Шаг 8. После завершения редактирования библиотеки типов нужно щелкнуть по кнопке **Refresh Implementation** (Обновить установку) и Delphi автоматически сгенерирует файл ruXmlMenuSrv_TLB.pas и модуль, который надо сохранить под именем u_XmlTreeSrv.pas.

Шаг 9. В файле ruXmlMenuSrv_TLB.pas ничего менять не нужно. Он предназначен для использования в других программах, обращающихся к нашему объекту автоматизации. Изучить его следует хотя бы для того, чтобы понять, какой объем работы выполнен автоматически. Полезно разобраться, как преобразуются типы данных из библиотеки типов в типы языка Delphi (листинг 16.3, в котором, для экономии места, удалены объемные комментарии).

Листинг 16.3. Файл ruXmlMenuSrv_TLB.pas

```
unit ruXmlMenuSrv_TLB;

{$TYPEDADDRESS OFF} // Unit must be compiled without type-checked pointers.
{$WARN SYMBOL_PLATFORM OFF}
{$WRITEABLECONST ON}
{$VARPROPSETTER ON}
interface

uses Windows, ActiveX, Classes, Graphics, StdVCL, Variants;


const
  ruXmlMenuSrvMajorVersion = 1;
  ruXmlMenuSrvMinorVersion = 0;
```

```
  LIBID_ruXmlMenuSrv: TGUID = '{D0B6EAD1-303C-42FB-9D63-4972F33F0529}';
  IID_IXmlTree: TGUID = '{1422AA9E-7F28-4D38-B7D2-FD37ADCB940A}';
  CLASS_XmlTree: TGUID = '{E9FD7EEA-2706-4CC8-A0C3-1667557D561C}';
type
  IXmlTree = interface;
  IXmlTreeDisp = dispinterface;
  XmlTree = IXmlTree;
  IXmlTree = interface(IDispatch)
    ['{1422AA9E-7F28-4D38-B7D2-FD37ADCB940A}']
    function  Execute(const Dlg_Caption: WideString;
                      const Xml_FileName: WideString;
                      const XML_RootDir: WideString;
                      const XML_ImagesDir: WideString;
                      const Tree_ViewedAttrib: WideString;
                      const Include_Attrib: WideString;
                      const Image_Attrib: WideString;
                      const Comment_Attrib: WideString;
                      const Required_Attrib: WideString;
                      const Ok_ButtonCaption: WideString;
                      const Cancel_ButtonCaption: WideString;
                      const Storage_IniFile: WideString;
                      const Res_IniFileName: WideString;
                      const Acad_AppString: WideString;
                      Multi_Select: WordBool;
                      Capture_ImageEnabled: WordBool;
                      Send_AcadEnabled: WordBool;
                      Last_Select: Integer): OleVariant; safecall;
  end;
 end;

  IXmlTreeDisp = dispinterface
    ['{1422AA9E-7F28-4D38-B7D2-FD37ADCB940A}']
    function  Execute(const Dlg_Caption: WideString;
                      const Xml_FileName: WideString;
                      const XML_RootDir: WideString;
                      const XML_ImagesDir: WideString;
                      const Tree_ViewedAttrib: WideString;
                      const Include_Attrib: WideString;
                      const Image_Attrib: WideString;
                      const Comment_Attrib: WideString;
                      const Required_Attrib: WideString;
                      const Ok_ButtonCaption: WideString;
                      const Cancel_ButtonCaption: WideString;
                      const Storage_IniFile: WideString;
                      const Res_IniFileName: WideString;
                      const Acad_AppString: WideString;
                      Multi_Select: WordBool;
                      Capture_ImageEnabled: WordBool;
                      Send_AcadEnabled: WordBool;
                      Last_Select: Integer): OleVariant; safecall;
                      OleVariant; dispid 19;
  end;
```

```
  CoXmlTree = class
    class function Create: IXmlTree;
    class function CreateRemote(const MachineName: string): IXmlTree;
  end;
implementation
uses ComObj;
class function CoXmlTree.Create: IXmlTree;
begin
  Result := CreateComObject(CLASS_XmlTree) as IXmlTree;
end;
class function CoXmlTree.CreateRemote(const MachineName: string): IXmlTree;
begin
  Result := CreateRemoteComObject(MachineName, CLASS_XmlTree) as IXmlTree;
end;
end.
```

Разбирать листинг у нас просто нет места, да это и не соответствует теме нашей книги. Обратим внимание на то, что в разделе `interface` появились присвоенные нашему объекту константы, такие как:

- `LIBID_ruXmlMenuSrv: TGUID = '{D0B6EAD1-303C-42FB-9D63-4972F33F0529}';`
- `IID_IXmlTree: TGUID = '{1422AA9E-7F28-4D38-B7D2-FD37ADCB940A}';`
- `CLASS_XmlTree: TGUID = '{E9FD7EEA-2706-4CC8-A0C3-1667557D561C}'` — типы, объявления функций и процедур (по числу свойств и методов).

В разделе `implementation` появилась реализация всего двух классов:

- `CoXmlTree.Create`;
- `CoXmlTree.CreateRemote`.

Почему все именно так — это темы толстых и умных книг по COM-технологиям. Пока ответим на напрашивающийся вопрос: "А где же реализация остальных объявленных процедур и функций?". Ответ простой — в модуле u_XmlTreeSrv.pas, который Delphi любезно нам приготовила и в который осталось только вписать сущую мелочь. Чем мы сейчас и займемся.

Шаг 10. Редактируем файл u_XmlTreeSrv.pas (листинг 16.4).

Листинг 16.4. Файл u_XmlTreeSrv.pas

```
unit u_XmlTreeSrv;
{
По традиции введенный нами текст выделен жирным шрифтом
}

{$WARN SYMBOL_PLATFORM OFF}

interface

uses
  ComObj, ActiveX, ruXmlMenuSrv_TLB, StdVcl, ComCtrls, JvComCtrls;
```

```
type
  TXmlTree = class(TAutoObject, IXmlTree)
  protected
function TXmlTree.Execute(const Dlg_Caption, Xml_FileName, XML_RootDir,
        XML_ImagesDir, Tree_ViewedAttrib, Include_Attrib, Image_Attrib,
        Comment_Attrib, Required_Attrib, Ok_ButtonCaption,
        Cancel_ButtonCaption, Storage_IniFile, Res_IniFileName,
        Acad_AppString: WideString; Multi_Select, Capture_ImageEnabled,
        Send_AcadEnabled: WordBool; Last_Select: Integer): OleVariant;
        safecall;
    { Protected declarations }
  end;

implementation

uses ComServ, frmXmlMenu, Forms, Controls, Dialogs, SysUtils, IniFiles;

var
  TreeXmlForm: TformXMLmenuShow;

function TXmlTree.Execute(const Dlg_Caption, Xml_FileName, XML_RootDir,
        XML_ImagesDir, Tree_ViewedAttrib, Include_Attrib, Image_Attrib,
        Comment_Attrib, Required_Attrib, Ok_ButtonCaption,
        Cancel_ButtonCaption, Storage_IniFile, Res_IniFileName,
        Acad_AppString: WideString; Multi_Select, Capture_ImageEnabled,
        Send_AcadEnabled: WordBool; Last_Select: Integer): OleVariant;
        safecall;
begin
  TreeXmlForm := TformXMLmenuShow.Create(Application);
  with TreeXmlForm do
  begin
    DlgCaption := Dlg_Caption;
    XmlFileName := ExpandFileName(Xml_FileName);
    XMLImagesDir := XML_ImagesDir;
    XMLRootDir := XML_RootDir;
    MultiSelect := Multi_Select;
    CaptureImageEnabled := Capture_ImageEnabled;
    SendAcadEnabled := Send_AcadEnabled;
    TreeViewedAttrib := Tree_ViewedAttrib;
    IncludeAttrib := Include_Attrib;
    ImageAttrib := Image_Attrib;
    CommentAttrib := Comment_Attrib;
    RequiredAttrib := Required_Attrib;
    OkButtonCaption := Ok_ButtonCaption;
    CancelButtonCaption := Cancel_ButtonCaption;
    LastSelect := Last_Select;
    StorageIniFile := ExpandFileName(Storage_IniFile);
    AcadAppString := Acad_AppString;
    ResIniFileName := ExpandFileName(Res_IniFileName);
```

```
    Result := (ShowModal = mrOk);
    Free;
  end;
end;

initialization
  TAutoObjectFactory.Create(ComServer, TXmlTree, Class_XmlTree,
    ciMultiInstance, tmApartment);
end.
```

Итак, мы видим, что реализация объекта автоматизации проста. Все получилось только немного занудно из-за того, что мы придумали целых 16 параметров для метода `Execute`.

Шаг 11. Компилируем приложение, нажав комбинацию клавиш <Ctrl>+<F9>. Если все правильно, появится файл ruXmlMenuSrv.dll. Это и есть желанная библиотека — сервер автоматизации. Мы можем переместить этот файл в каталог .ru\CAD\Bin, но это сделаем по достижении полной готовности, а пока испытаем его по месту временной регистрации. Но регистрации нужно еще выполнить.

Шаг 12. Выполняем регистрацию сервера из командной строки

```
Regsvr32 ruXmlMenuSrv.dll
```

Так мы будем делать при перемещении библиотеки на другое место или при установке на другой компьютер. Сейчас, пока мы находимся в среде Delphi, мы можем из меню **View | Type Library** (Вид | Библиотека типов) вызвать редактор библиотеки типов (именно так он всегда и вызывается для корректировки библиотеки) и щелкнуть по кнопке **Register Type Library** (Регистрация библиотеки типов). Возможно, проект перекомпилируется, и библиотека будет зарегистрирована. В результате в реестре Windows появятся соответствующие ключи, например:

```
[HKEY_CLASSES_ROOT\ruXmlMenuSrv.XmlTree\Clsid]
@="{E9FD7EEA-2706-4CC8-A0C3-1667557D561C}"
```

Теперь наш сервер готов к работе.

Тестовое приложение для проверки COM-сервера

Напишем небольшое тестовое приложение для проверки работы, чтобы потом, при разработке LISP-функций, точно знать, что сервер работает правильно. "Разжевывать" тестовое приложение мы не будем. Это простая форма со множеством компонентов для редактирования всех свойств и тремя кнопками для проверки метода.

Из исходного текста тестового приложения мы приведем только фрагменты, где производится вызов методов сервера (листинг 16.5).

Листинг 16.5. Фрагмент файла frmTestSrv.pas

```
unit frmTestSrvLT;

interface

uses
  Windows, Messages, SysUtils, {Variants,} Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Mask, JvToolEdit, ruXmlMenuSrv_TLB;
```

```
type
  TForm1 = class(TForm)
    JvFilenameEdit1: TJvFilenameEdit;
    Memo1: TMemo;
    CheckBoxCapture: TCheckBox;
    CheckBoxMultiSelect: TCheckBox;
    CheckBoxSendAcad: TCheckBox;
    JvDirectoryEdit1: TJvDirectoryEdit;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    Label8: TLabel;
    Label9: TLabel;
    Label10: TLabel;
    Label11: TLabel;
    Button2: TButton;
    Edit1: TEdit;
    EditImage: TEdit;
    EditName: TEdit;
    EditInclude: TEdit;
    EditComment: TEdit;
    EditRequired: TEdit;
    EditOk: TEdit;
    EditCancel: TEdit;
    procedure FormCreate(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
    ruXmlTree: IXmlTree;

  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}
uses comobj;

procedure TForm1.FormCreate(Sender: TObject);
begin
{
COM-объект создается при генерации формы. Единственную строку можно просто
скопировать из ruXmlMenuSrv_TLB.pas
}
```

```
  ruXmlTree := CreateComObject(CLASS_XmlTree) as IXmlTree;
end;

procedure TForm1.Button2Click(Sender: TObject);
var
  XmlFName,
    StorageFname, ResultFName: string;
  ResDial: OleVariant;
begin
  Memo1.Lines.Clear;
  XmlFName := JvFilenameEdit1.FileName;
  if ((XMLFname <> '') and FileExists(XMLFname)) then
  begin
    ResultFName := ChangeFileExt(XmlFname, '.out');
    StorageFname := ChangeFileExt(XmlFname, '.ini');
    ResDial := ruXmlTree.Execute('Выбор из меню', XmlFName,
      ExtractFilePath(ExpandFileName(XmlFName)),
      'c:\.ru\CAD\All Users\XML\IMAGES\',
      EditName.Text,
      EditInclude.Text, EditImage.Text,
      EditComment.Text, EditRequired.Text,
      EditOk.Text, EditCancel.Text,
      StorageFname, ResultFName,
      Edit1.Text, CheckBoxMultiSelect.Checked,
      CheckBoxCapture.Checked, CheckBoxSendAcad.Checked, 1);
    if ResDial then
      Memo1.Lines.LoadFromFile(ResultFName)
    else
      ShowMessage('Отменено');
  end
  else
    ShowMessage('Не найден ' + XMLFname);

  end;
end.
```

Запуск тестового приложения подтвердил правильность работы изготовленного сервера автоматизации. Теперь можно переходить к попыткам достучаться до него из Visual LISP.

Напишем функции для вызова приложения из Visual LISP

Сначала напишем низкоуровневую функцию выбора из XML-документа. Эта функция потребует максимального количества параметров.

Листинг 16.6. Функция _ru-xml-tree-select

```
(defun _ru-xml-tree-select (dlg_caption full_xml_file_name
                        att_names_list display_att comment_att
```

```
                        required_att is_multiselect is_capture
                        is_send_acad
                        / ini_file lastselected last_param
                        result srv res_srv)
;;; Локальная функция
(defun _make-att-list-result-single (att_list ini_file section / result i
                                                          section_list)
;| Формирование списка атрибут-значение по результатам выбора одного узла
   Если att_list nil, читать все атрибуты
|;
    (setq result '())
    (if att_list
      (foreach att_name att_list
        (setq result (cons (cons att_name
                                 (ru-ini-read
                                   ini_file
                                   section
                                   att_name
                                   ""
                                 );_ end of ru-ini-read
                           );_ end of cons
                           result
                     );_ end of cons
       );_ end of setq
     );_ end of foreach
      (progn
        (setq section_list (ru-ini-read-list ini_file section)
              i 0)
        (foreach att_name (car section_list)
          (setq result (cons (cons att_name
                                   (nth i (cadr section_list))
                             );_ end of cons
                             result
                       );_ end of cons
        );_ end of setq
          (setq i (1+ i))
      );_ end of foreach
    );_ end of progn
   );_ end of if
    (reverse result)
 );_ end of defun
;;; Локальная функция
  (defun _make-att-list-result-multi (att_list ini_file /
                                      count item list_item result)
;| Формирование списка атрибут-значение по результатам выбора нескольких узлов
   Если att_list nil, читать все атрибуты
|;
   (setq result '()
         count  (atoi (ru-ini-read
                        ini_file
```

```
                        "multiselect"
                        "SelectionCount"
                        "0"
                      );_ end of ru-ini-read
                );_ end of atoi
          item   1
    );_ end of setq
     (repeat count
       (if att_list
         (progn
           (setq list_item '())
           (foreach att_name att_list
             (setq list_item (cons (cons att_name
                                         (ru-ini-read
                                           ini_file
                                           (strcat "item" (itoa item))
                                           att_name
                                           ""
                                         );_ end of ru-ini-read
                                   );_ end of cons
                                    list_item
                             );_ end of cons
           );_ end of setq
         );_ end of foreach
       );_ end of progn
        (setq list_item (_make-att-list-result-single
                          nil
                          ini_file
                          (strcat "item" (itoa item))
                        );_ end of _make-att-list-result-single
       );_ end of setq
     );_ end of if
      (setq item   (1+ item)
            result (cons (reverse list_item) result)
     );_ end of setq
   );_ end of repeat
    (reverse result)
 );_ end of defun
;;;### ГЛАВНАЯ ФУНКЦИЯ ###
  (setq result nil
        last_param
                        (strcat full_xml_file_name "-lastselected")
        is_capture
                        (ru-conv-value-to-wordbool is_capture)
        is_multiselect
                        (ru-conv-value-to-wordbool is_multiselect)
        is_send_acad
                        (ru-conv-value-to-wordbool is_send_acad)
        lastselected
                        (ru-user-read-last-param
```

```
                        last_param
                        "0"
                      );_ end of ru-user-read-last-param
       ini_file       (ru-file-tmp "~XmlTree.ini")
 );_ end of setq
  (vl-file-delete ini_file)
  (if (setq srv (vlax-get-or-create-object "ruXmlMenuSrv.XmlTree"))
    (progn
      (setq res_srv (vlax-invoke-method
                      srv
                      "Execute"
                      dlg_caption
                      full_xml_file_name
                      (ru-file-path full_xml_file_name)
                      (ru-dirs-menu-xml-images)
                      display_att
                      "include"
                      "image"
                      comment_att
                      required_att
                      "Выполнить"
                      "Закрыть"
                      (ru-file-current-user "XmlMenuStorage.ini")
                      ini_file
                      "AutoCAD.Application"
                      is_multiselect
                      is_capture
                      is_send_acad
                      lastselected
                    );_ end of vlax-invoke
    );_ end of setq
     (if (= (vlax-variant-value res_srv) :vlax-true)
       (progn
         (if (ru-conv-value-to-bool is_multiselect)
           (setq result (_make-att-list-result-multi att_names_list
                         ini_file))
           (setq result (_make-att-list-result-single att_names_list
                         ini_file "singleselect"))
        );_ end of if
         (ru-user-write-last-param last_param (ru-ini-read
            ini_file "result" "lastselected" "0"))
         (if is_send_acad
           (setq result T)
        );_ end of if
      );_ end of progn
    );_ end of if
   );_ end of progn
    (princ "\nНе удалось запустить XMLtree")
 );_ end of if
  result
);_ end of defun
```

Сделаем необходимые пояснения. Параметры функции:

- `dlg_caption` — заголовок диалога;
- `full_xml_file` — имя XML-файла;
- `att_names_list` — список имен атрибутов, значения которых нужно вернуть, или `NIL`, если требуется вернуть значения всех атрибутов;
- `display_att` — имя атрибута, значение которого нужно показывать вместо имени элемента;
- `comment_att` — имя атрибута, значение которого нужно показывать вместо текста элемента, если текст отсутствует (краткие пояснения удобнее размещать в атрибуте);
- `required_att` — имя атрибута, отсутствие которого или пустая строка в котором блокируют кнопку **OK**;
- `is_multiselect` — возможность выбора нескольких пунктов;
- `is_capture` — разрешается ли замена иллюстраций;
- `is_send_acad` — разрешается ли прямая отправка макроса в командную строку.

Эта функция является наиболее гибкой. Если произведен выбор из XML-документа, функция вернет результаты:

```
((имя_атрибута . значение_атрибута) ...)
```

при выборе одного элемента и

```
 (
  ((имя_атрибута . значение_атрибута) ...)
  ((имя_атрибута . значение_атрибута) ...)
)
```

при выборе нескольких элементов.

Такую функцию сложно использовать в типовых ситуациях, поэтому делаем для нее функции-оболочки (листинги 16.7—16.11).

Листинг 16.7. Функция `ru-xml-select-macro`

```
(defun ru-xml-select-macro (full_xml_file_name dlg_caption
                            / macro lst result res_check)
;;; выбор  из дерева с возвратом списка данных
;;; Параметров только два, остальные - типовые
;;; Возвращает строку "macro", проверенную на ошибки или nil
  (if (setq lst (_ru-xml-tree-select dlg_caption full_xml_file_name
                (list "name" "macro") "name" "comment" "macro" nil t
                nil
              );_ end of _ru-xml-tree-select
    );_ end of setq
    (progn
;;; Отбрасываем весь мусор в начале и конце строки
      (setq macro (ru-string-rem-all
                    (car (ru-list-massoc "macro" lst))
                    '(" " "\t" "[" "]" ";" "\"")
```

```
                );_ end of ru-string-rem-all
     );_ end of setq
;;; Проверяем синтаксис строки
      (if (= (setq res_check (ru-lsp-check macro)) "")
        (setq result macro)
        (progn
          (ru-msg-alert (strcat "Ошибка в XML-меню:\n" res_check))
          (setq result nil)
       );_ end of progn
     );_ end of if
   );_ end of progn
 );_ end of if
  result
);_ end of defun
```

Эта функция только позволяла выбрать элемент и проверить результат, а `ru-xml-eval` еще и выполняет полученный макрос, и делает все это в цикле.

Листинг 16.8. Функция `ru-xml-eval`

```
(defun ru-xml-eval (full_xml_file_name dlg_caption / macro)
;;; выбор и выполнение  пункта XML-меню
  (while (setq macro (ru-xml-select-macro full_xml_file_name
                       dlg_caption))
    (ru-lsp-eval macro)
 );_ end of while
);_ end of defun
```

Но и это еще не все. Конечным продуктом является функция `ru-xml-pop-mnu`, записываемая в MNU-файл. Этой функции передается не полное, а короткое имя XML-документа. Теперь в нашем меню нигде не будет абсолютных имен файлов, только относительные.

Листинг 16.9. Функция `ru-xml-pop-mnu`

```
(defun ru-xml-pop-mnu (relative_base_xml_file_name dlg_caption)
;;; Вызов XML-меню из меню AutoCAD
;;; Пример: (ru-xml-pop-mnu "tabl_gp" "Таблицы для ГЕНПЛАНА")
  (ru-xml-eval
    (ru-file-set-ext
        (ru-file-menu-xml relative_base_xml_file_name) ".xml")
     dlg_caption
 );_ end of ru-xml-eval
  (princ)
);_ end of defun
```

Кроме вызова программ, у нас часто будет производиться выбор из XML-документов различных опций. В такой ситуации используются разные атрибуты и нужно иметь специальную функцию (листинг 16.10).

Листинг 16.10. Функция `ru-xml-get-att-list`

```
(defun ru-xml-get-att-list (dlg_caption full_xml_file_name att_names_lst
required_att)
 (_ru-xml-tree-select dlg_caption full_xml_file_name att_names_lst
                      "name" "comment" required_att nil T nil)
)
```

Пример:

```
(ru-xml-get-att-list "Выбор диаметра"
                     (ru-file-menu-xml "options\\flange.xml")
   (list "name" "du" "dfl" "db" "sf" "dv" "lv" "ln" "dotv" "n")
   "du"
);_ end of ru-xml-get-att-list
```

Может вернуть

```
(("name" . "Фланец Ду 200")("du" . "200.0")("dfl" . "335.0")
   ("db" . "295.0")("sf" . "21.0")("dv" . "268.0")("lv" . "3.0")
   ("ln" . "58.0") ("dotv" . "22")("n" . "12"))
```

И, наконец, напишем последнюю функцию (листинг 16.11), требуемую для совместимости со старыми программами системы BestIA, в которой для хранения данных использовалась одна строка и в которой данные разделялись точкой с запятой. Это были файлы формата BTR, преобразованные теперь в XML. Строка данных содержится в атрибуте с именем `SDATA`.

Листинг 16.11. Функция `ru-xml-get-sdata`

```
(defun ru-xml-get-sdata (dlg_caption full_xml_file_name)
 (_ru-xml-tree-select dlg_caption full_xml_file_name (list "sdata")
                      "name" "comment" "sdata" nil T nil)
)
```

Работа с XML-таблицами

В рассматривавшихся ранее примерах основной упор делался на иерархию информации в XML-документах. Каждый элемент мог иметь разные атрибуты, элементы могли иметь разные имена, хотя в конкретной программной системе желательно придерживаться стандартных имен атрибутов (у нас это были `"name"`, `"image"`, `"macro"`, `"comment"` и `"sdata"`).

Существует еще одна область применения XML — хранение таблиц с жестко заданной структурой. Традиционно для таких целей используются реляционные базы данных, в которых информация содержится в одной или нескольких таблицах, связанных между собой различными отношениями. В теорию "настоящих" баз данных мы углубляться не будем. Имеется множество применений и для простых таблиц ("плоских" файлов), используемых локально, без разделения доступа с другими пользователями. Типичный пример — ведомость чертежей основного комплекта для конкретного проекта, да и большинство других таблиц, входящих в состав общих данных по рабочим чертежам.

Такие таблицы можно просто нарисовать в чертеже и заполнять их простым текстом. Чаще всего так и делается. Некоторые таблицы можно заполнять автоматически на основании какой-то информации, содержащейся в рисунке. В своей системе мы таких ухищрений делать не будем. Иногда пользователи вставляют внутрь рисунка таблицу, сделанную в Microsoft Excel. Дело это весьма ненадежное, любительского уровня. Работоспособность "решений Microsoft", конечно, будет подтверждена, но только до поры, когда рисунок не будет перемещен в другое место.

Постановка задачи

Давайте отработаем простое и надежное решение — редактирование таблиц в формате XML. Сначала продумаем, что и как мы должны делать с таблицами.

Во-первых, таблицу нужно нарисовать в системе AutoCAD и разграфить. Это делается легко и просто с помощью наших функций.

Во-вторых, содержательную часть таблицы нужно хранить на диске для возможности добавлений и изменений. Здесь имеется множество вариантов. Можно придумать свой текстовый формат, можно хранить в полустандартных форматах с фиксированной шириной колонок и с разделением колонок специальными символами, а можно хранить данные в распространенных форматах баз данных DBF или MDB. LISP-программисты любят хранить данные в текстовых файлах в виде списков, которые удобно читать из LISP-программ.

В-третьих, файл с содержимым таблицы нужно редактировать. Текстовые файлы, конечно, можно редактировать обычным текстовым редактором, но тут велика вероятность ошибки — не тот символ, не на том месте и всю информацию можно потерять. Можно написать специальную программу для редактирования конкретной таблицы. Примерами таких программ заполнены все книги по работе с базами данных. Но если мы имеем сотни таблиц, то понадобятся и сотни программ. Решение заключается в том, чтобы отображать таблицы в универсальной табличной форме. В различных системах программирования и работы с базами данных имеются соответствующие средства. Мы рассматриваем программирование в Delphi и будем придерживаться терминов и технологий фирмы Borland.

Если таблица находится в базе данных любого стандартного формата, то все решается просто:

- на форму помещается компонент, имеющий в "предках" класс `TDataSet`, например, `TTable`, компонент `DataSource` и какая-нибудь "сетка" класса `TdbGrid` и, факультативно, компонент `DbNavigator`;
- после установки соответствующих свойств уже на этапе разработки в сетке немедленно отобразится таблица;
- после компиляции и запуска программы мы имеем редактор таблицы, который умеет многое — передвигаться по таблице, редактировать данные, сохранять их на диск.

Компонент `Table` сам или через посредников в виде каких-либо "движков баз данных" (DataBase Engine) читает и пишет информацию, `DataSource` переправляет данные в визуальные компоненты, а `DbGrid` отображает информацию. Но не все так просто, как кажется.

Возникают вопросы:

- Как первоначально создать таблицу?
- Как ее красиво и понятно отобразить — в `DbGrid` отображаются короткие, обычно нерусские имена полей?
- И наконец, животрепещущий вопрос, ежедневно задаваемый на форумах, — как избавиться от BDE, ADO и прочих "движков" баз данных. Хочется, чтобы все было в "моем EXE-шнике".

Самое первое "яйцо", из которого будут выводиться "курицы", т. е. саму таблицу, создают обычно с помощью внешних программ. Например, это может быть система Microsoft Access или другие "десктопы" и "эксперты". Часто базы данных и таблицы создают с помощью языка SQL. Все это требует определенных знаний и явно не рассчитано на наших любимых "теток". База данных и таблицы могут создаваться и во время исполнения, но для этого необходимо где-то хранить структуру таблицы, т. е. имена полей, типы данных и прочую информацию. Многие программисты хранят структуры в таких же таблицах, мы обычно используем для таких целей INI-файлы, для создания которых не нужен никакой специальный инструмент.

Отобразить таблицу "как надо" довольно просто. Надо только использовать подходящий компонент. Многие из них умеют отображать длинные многострочные заголовки столбцов, удобно показывать данные (с "чекбоксами", календариками и прочими милыми прелестями). Очень хорош компонент `DbGridEh`, созданный блестящим программистом Дмитрием Большаковым и, что важно, доступный бесплатно пользователям из exUSSR.

Однако свойства столбцов сетки надо также установить. Для конкретной таблицы это можно сделать в design-time (во время разработки), а для "обезличенной" нужно делать в run-time (во время исполнения приложения), для чего дополнительные свойства колонок (как минимум, заголовок) надо откуда-то прочитать. Некоторые СУБД, например Microsoft Access, хранят эту информацию внутри своей базы данных, в служебных таблицах, доступ к которым из других программ весьма затруднен. Сама же таблица внутри содержит только имена полей, но не их заголовки.

В современных программах доступ к базам данных обычно производится через типовые механизмы. Длительное время использовался *BDE* (Borland DataBase Engine), теперь широко применяется технология Microsoft ADO. С одной стороны, это хорошо, с другой стороны на компьютере пользователя должен быть установлен соответствующий "движок", а это дополнительные мегабайты в довесок к вашей программе. Большинство начинающих программистов стремится избавиться от "движков", большинство профессионалов советуют им этого не делать, а использовать стандартные средства. Иногда "профи" поражены известной болезнью под названием "специализм" и советуют, даже для заведомо локальных данных с мизерным объемом информации, устанавливать настоящие СУБД, такие как Interbase, MS SQL или даже Oracle.

Имеются много СУБД, не требующих установки у клиентов дополнительных компонентов, но недостаток их в том, что используются нестандартные форматы данных.

В-четвертых, информацию из таблицы на диске нужно как-то переправить в AutoCAD и заполнить нарисованную форму. Для этого или наша LISP-программа должна уметь напрямую читать файл таблицы, или обмен должен производиться через промежуточный текстовый формат, доступный для чтения из LISP.

XML-таблицы

Разумеется, многие из упомянутых проблем давно решены. Одним из очень хороших решений является использование так называемых "таблиц в памяти". Пожалуй, лучший из компонентов — `kbmMemTable` Кима Мадсена, позволяющий работать с текстовыми файлами формата CSV как с обычным DataSet. Этот компонент имеет множество возможностей, но не свободен и от недостатков. В частности, структура таблицы может храниться в самом файле, в собственном жестком формате. А это уже делает формат нестандартным. Да и дополнительные, нужные нам, свойства туда не записать.

Решение всех (ну, почти всех) проблем, заключается в использовании языка XML. XML широко используется для обмена информацией между различными СУБД. Существуют даже очень мощные СУБД, полностью работающие с XML. Фирма Borland разработала великолепный компонент `ClientDataSet`, для работы которого нужен только файл midas.dll. Эта технология была создана для того, чтобы подключившись к удаленному серверу баз данных, получить информацию, сохранить любой набор данных локально, отключиться от удаленного сервера, работать локально, а потом, при необходимости, синхронизировать данные. Программисты сразу сообразили, что это можно прекрасно использовать и для работы с локальными данными.

Microsoft Access 2002 также использует XML для обмена информацией с другими СУБД. Недостаток технологий крупных фирм в том, что форматы несовместимы между собой. Кроме того, из-за слишком больших возможностей форматы файлов очень сложны для использования в простых приложениях. И, разумеется, отвратительная работа с русскими символами. Borland `ClientDataSet` сохраняет их в виде номеров кодов. Помимо увеличения размера файлов это делает их нечитаемыми в обычном текстовом редакторе.

В результате длительных поисков мы нашли компонент, полностью удовлетворяющий нашим требованиям. Ян Верховен, автор уже использованного нами `janXMLParser2`, разработал превосходный компонент `janXMLDataSet2`. Этот компонент также умеет делать все, что надо, и не умеет делать ничего лишнего. Разбирать текст компонента мы не будем, он доступен для бесплатного использования на сайте **jansfreeware.com** в составе пакета EasyXML. Перейдем сразу к созданию таблиц.

Создание XML-таблицы

Первую таблицу создадим в текстовом редакторе (листинг 16.12). Пусть это будет таблица основных показателей по чертежам водопровода и канализации. Ее мы возьмем потому, что в этой таблице необходимо показывать сложный заголовок с несколькими столбцами под одной шапкой. В этом листинге полужирными символами мы выделим то, что не предусмотрено автором `janXmlDataSet2`. Это наглядно демонстрирует преимущества языка XML — возможность введения дополнительных атрибутов.

Листинг 16.12. Файл 'Основные показатели ВК.xml'

```
<?xml version='1.0' encoding='windows-1251' ?>
<table>
  <fields>
    <система size='100' type='string'
      displayname='Наименование системы' columnwidth='120'
```

```
 picklist='В1 - хоз.-питьевой|В2 – производственный|Т3 - гор.водоснабжение'/>
    <напор size='9' type='float'
      displayname='Потребный напор на вводе, м' columnwidth='80'/>
    <м3_сутки size='9' type='float'
      displayname='Расчетный расход| м3/сут' columnwidth='50'
      displayformat='0.000'/>
    <м3_час size='9' type='float'
       displayname='Расчетный расход| м3/час' columnwidth='50'
       displayformat='0.000'/>
    <л_сек size='9' type='float'
        displayname='Расчетный расход| л/с' columnwidth='50'
        displayformat='0.000'/>
    <пожарный size='9' type='float'
       displayname='Расчетный расход|При пожаре л/с' columnwidth='50'
       displayformat='0.000'/>
    <квт size='9' type='float'
        displayname='Установленная мощность электродвигателей, кВт'
        columnwidth='120'/>
    <прим size='100' type='string'
      displayname='Примечание' columnwidth='100'/>
  </fields>
  <rows>
<row>
      <система>В1 - хоз.-питьевой</система>
      <напор>12</напор>
      <м3_сутки>12</м3_сутки>
      <м3_час>2.2</м3_час>
      <л_сек>0.25</л_сек>
      <пожарный>10</пожарный>
      <квт>24</квт>
      <прим/>
    </row>
    <row>
      <система>Т3 - гор.водоснабжение</система>
      <напор>15</напор>
      <м3_сутки>4</м3_сутки>
      <м3_час>0.5</м3_час>
      <л_сек>0.2</л_сек>
      <пожарный>0</пожарный>
      <квт>0</квт>
      <прим/>
    </row>
    <row>
      <система>В2 - производственный</система>
      <напор>10</напор>
      <м3_сутки>10</м3_сутки>
      <м3_час>1</м3_час>
      <л_сек>0.25</л_сек>
      <пожарный>0</пожарный>
```

```
      <квт>0</квт>
      <прим/>
    </row>  </rows>
  <table/>
</table>
```

Формат файла достаточно простой. Особенность его в том, что мы уже должны соблюдать обязательные правила. Корневой элемент документа имеет обязательный элемент `table`, в него вложены обязательные элементы `fields` (определения полей) и `rows` (собственно, данные).

В элемент `fields` должны быть вложены не менее одного элемента, определяющего поля (колонки) таблицы. Поле определяется элементом, имя которого будет именем поля в таблице. Поле должно иметь обязательные атрибуты `size` — размер данных в поле и `type` — тип данных в поле (`string`, `integer`, `float`, `datetime`, `boolean` и `memo`). Хотя в XML все данные хранятся в текстовом виде, они должны правильно преобразовываться в соответствующий тип. Вот и все стандартные правила, соблюдение которых обязательно. По возможностям такой формат примерно соответствует формату DBF. Индексы не поддерживаются и не нужны.

Теперь займемся добавлением собственных свойств полей. Дополнительные свойства мы задаем через атрибуты. Для первого раза мы введем следующие свойства:

- `displayname` — свойство, соответствующее свойству `DisplayName` у полей баз данных, т. е. текст, отображающийся в заголовках `DbGrid` вместо имени поля, например `'Наименование системы'`;
- `columnwidth` — ширина колонки `DbGrid` в пикселах;
- `picklist` — строка, содержащая список подстрок, разделенных символом "|" — мини-словарь, облегчающий ввод данных, например `'Иванов|Петров|Сидоров'`;
- `displayformat` — строка-шаблон вывода данных в стандартном формате, например шаблон `'0.000'` обеспечит вывод чисел с тремя знаками после запятой.

Дополнительные атрибуты были выделены в листинге полужирным шрифтом.

Программа редактирования XML-таблиц

Теперь быстренько сделаем редактор таблиц (рис. 16.12). Создаем в Borland Delphi новый проект, размещаем на форме несколько компонентов и сохраняем форму в файле frmXmlTableLite (листинг 16.13).

Листинг 16.13. Форма frmXmlTableLite.dfm (сокращено)

```
object frmGrid: TfrmGrid
  Caption = 'Редактор таблицы'
  OnClose = FormClose
  OnShow = FormShow
  object PanelAll: TPanel
    object DBGrid: TDBGridEh
      Align = alClient
      AllowedSelections = []
```

```
      DataSource = DataSourceXML
      DrawMemoText = True
      EditActions = [geaCutEh, geaCopyEh, geaPasteEh, geaDeleteEh,
                     geaSelectAllEh]
      Flat = True
      Options = [dgEditing, dgTitles, dgIndicator, dgColumnResize,
                 dgColLines, dgRowLines, dgTabs, dgConfirmDelete,
                 dgCancelOnExit, dgMultiSelect]
      OptionsEh = [dghFixed3D, dghResizeWholeRightPart,
                   dghHighlightFocus, dghClearSelection,
                   dghFitRowHeightToText, dghAutoSortMarking,
                   dghMultiSortMarking, dghTraceColSizing]
      RowHeight = 2
      RowLines = 1
      RowSizingAllowed = True
      SumList.Active = True
      UseMultiTitle = True
      OnCellClick = DBGridCellClick
    end
    object Panel2: TPanel
      object btnOk: TButton
        Caption = 'OK'
        Default = True
        ModalResult = 1
        OnClick = btnOkClick
      end
      object btnCancel: TButton
        Cancel = True
        Caption = 'Отмена'
        ModalResult = 2
        OnClick = btnCancelClick
      end
      object DBNavigator1: TDBNavigator
        DataSource = DataSourceXML
        VisibleButtons = [nbFirst, nbPrior, nbNext, nbLast, nbInsert,
                          nbDelete, nbEdit, nbPost, nbCancel]
      end
    end
  end
  object DataSourceXML: TDataSource
    DataSet = XMLDataSet
  end
  object XMLDataSet: TjanXMLDataSet2
    ReadOnly = False
  end
  object JvFormPlacement1: TJvFormPlacement
  end
end
```

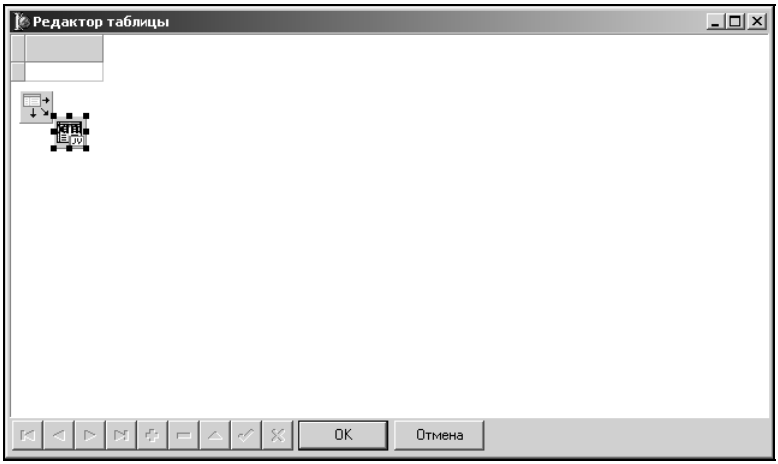


Рис. 16.12. Разработка редактора таблиц

Важнейшие компоненты и их свойства выделены полужирным шрифтом. Особо обращаем внимание на использование в качестве "сетки" компонента `DbGridEh`, т. к. именно этот компонент решит многие задачи без написания дополнительного кода. Свойства `DbGrid`, `DataSourceXML` и `XMLDataSet` настроены так, что в `DbGrid` будут отображаться строки и колонки, соответствующие значениям записей и полей в XMLDataSet.

Текст модуля с подробными комментариями показан в листинге 16.14.

Листинг 16.14. Модуль frmXmlTableLite.dfm

```
unit frmXmlTableLite;
interface

uses
  ShellAPI, Windows, Messages, SysUtils, Classes, Graphics, Controls,
  Forms, Dialogs, ComCtrls, StdCtrls, ExtCtrls, ToolWin, Db,
 janXMLDataSet2, DBCtrls, DBGridEh, Grids, Mask, JvToolEdit,
 BaseDataset, JvPlacemnt;

type
  TfrmGrid = class(TForm)
    DataSourceXML: TDataSource;
    PanelAll: TPanel;
    Panel2: TPanel;
    XMLDataSet: TjanXMLDataSet2;
    btnOk: TButton;
    btnCancel: TButton;
    DBGrid: TDBGridEh;
    DBNavigator1: TDBNavigator;
    JvFormPlacement1: TJvFormPlacement;
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
```

```
    procedure DBGridCellClick(Column: TColumnEh);
    function GetFieldAttribute(FieldName, Attribute: string): string;
    procedure btnCancelClick(Sender: TObject);
    procedure btnOkClick(Sender: TObject);
    procedure FormShow(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
    MultiSelect: boolean;
    XmlFileName, ResultFileName, StorageFileName,
    OkCaption,CancelCaption: string;
    procedure InitGrid;
  end;

var
  frmGrid: TfrmGrid;
implementation

{$R *.DFM}

uses
  DBGridEhImpExp, agStrUtils, dbGrids, janstrings, janXMLParser2,
  MemoEditU;

procedure TfrmGrid.FormClose(Sender: TObject; var Action: TCloseAction);
begin
{
При закрытии формы сохраняем ее размер и положение в StorageFileName
}
  JvFormPlacement1.SaveFormPlacement;
  XMLDataSet.Close;
end;

procedure TfrmGrid.DBGridCellClick(Column: TColumnEh);
var
  mf: TMemoField;
  tmp: string;
begin
{
Обработчик события двойного щелчка по ячейке сетки. У нас может быть и мемо-поле
с большим текстом. Для редактирования такого текста открывается специальная форма
с текстовым редактором
}
  if Assigned(dbgrid.SelectedField) and
     (dbgrid.SelectedField is TMemoField)
    then
  begin
{
Если выделенное поле имеет тип мемо, создаем форму
}
```

```
    MemoEditf := TMemoEditF.Create(Application);
    mf := TMemoField(Column.Field);
    tmp := mf.Value;
{
Текущее значение поля загружается в редактор
}
    MemoEditf.SynEdit1.lines.text := tmp;
    if MemoEditF.ShowModal = mrOK then
    begin
      XMLDataSet.edit;
{
Если редактирование закончилось нажатием OK, сохраняем в поле текст из редактора
}
      mf.value := MemoEditf.SynEdit1.lines.text;
{
и записываем изменения на диск
}
      XMLDataSet.post;
    end;
    MemoEditf.Free;
  end;
end;

function TfrmGrid.GetFieldAttribute(FieldName, Attribute: string): string;
{
Важная функция получения значения атрибута по имени поля
}
var
  xn: TjanXMLNode2;
begin
  result := FieldName;
{
В XMLDataSet.xfields содержатся определения полей. Если поля не определены, выходим
досрочно
}
  if XMLDataSet.xfields = nil then
    exit;
{
Ищем определение поля с заданным именем
}
  xn := XMLDataSet.xfields.getChildByName(FieldName);
  if xn = nil then
    exit;
{
Возвращаем значение атрибута
}
  result := xn.attribute[Attribute];
end;
```

```
procedure TfrmGrid.InitGrid;
{
Установка свойств DbGrid
}
var
  ColumnWidth, i, c: integer;
  DisplayName: string;
  Hidden: boolean;
  DisplayFormat, EditMask, PickListString: string;
  PickList: TStrings;
begin
{
Устанавливаем заданные по смыслу надписи на кнопках
}
  btnOk.Caption := OkCaption;
  btnCancel.Caption := CancelCaption;
  if FileExists(XMLFileName) then
  begin
    try
{
Открываем документ
}
      XMLDataSet.close;
      XMLDataSet.XMLFile := XMLFileName;
      XMLDataSet.Open;
{
Устанавливаем дополнительную опцию DbGrid, если задан выбор всех строк
}

      if MultiSelect then
        DBGrid.Options := DBGrid.Options + [dgMultiSelect];
{
Для каждого поля изучаем нестандартные атрибуты и устанавливаем свойства DbGrid
}
c := XMLDataSet.FieldCount;
      if c > 0 then
        for i := 0 to c - 1 do
        begin
{
Ищем атрибут displayname
}
          DisplayName := GetFieldAttribute(XMLDataSet.FieldDefs[i].Name,
            'displayname');
          if DisplayName <> '' then
{
Если он найден, заголовок колонки устанавливаем по значению этого атрибута (иначе
в заголовке останется имя поля)
}
            DBGrid.Columns[i].Title.Caption := DisplayName;
```

```
{
Маска редактирования. О ней мы не упоминали, но возможность предусмотрели
}

          EditMask := GetFieldAttribute(XMLDataSet.FieldDefs[i].Name,
            'editmask');
          if EditMask <> '' then
            DBGrid.Columns[i].EditMask := EditMask;
{
Формат отображения. Важен для числовых полей, иначе все числа будут выглядеть
по-разному
}

          DisplayFormat :=GetFieldAttribute(XMLDataSet.FieldDefs[i].Name,
            'displayformat');
          if DisplayFormat <> '' then
            DBGrid.Columns[i].DisplayFormat := DisplayFormat;
{
Ширина колонки сетки. Ширину можно изменять во время работы, но есть возможность
задать в атрибуте
}

          ColumnWidth :=
            strtointdef(GetFieldAttribute(XMLDataSet.FieldDefs[i].Name,
            'columnwidth'), 0);
          if ColumnWidth > 0 then
            DBGrid.Columns[i].Width := ColumnWidth;
{
Некоторые колонки можно сделать невидимыми и не отображать в сетке
}

          Hidden := strtobool(GetFieldAttribute(XMLDataSet.FieldDefs[i].Name,
            'hidden'));
          DBGrid.Columns[i].Visible := not Hidden;
{
Мини-словарик, позволяющий облегчить ввод путем выбора из раскрывающегося списка
}

          PickListString := GetFieldAttribute(XMLDataSet.FieldDefs[i].Name,
            'picklist');
          if PickListString <> '' then
          begin
            PickList := TStringList.Create;
            StringToList(PickListString, PickList, ['|']);
            DBGrid.Columns[i].PickList := PickList;
            PickList.Free;
          end;
   end;
```

```
    except
      Application.MessageBox(PChar('Не могу открыть Файл ' + XMLFileName),
        'Ошибка', MB_OK + MB_ICONHAND + MB_DEFBUTTON1 + MB_APPLMODAL);
    end;
  end
  else
    Application.MessageBox(PChar('Не найден ' + XMLFileName),
      'Ошибка', MB_OK + MB_ICONHAND + MB_DEFBUTTON1 + MB_APPLMODAL);

end;

procedure TfrmGrid.btnCancelClick(Sender: TObject);
begin
  Close;
end;

procedure TfrmGrid.btnOkClick(Sender: TObject);
{ Возврат результатов в удобном виде, чтобы вызывающая программа не
 анализировала XML. Вызывать можно не только из LISP, поэтому
 пишем в CSV-файл}
var
{
DbGridEh имеет специальный класс для экспорта, его мы и используем
}
  ExpClass: TDBGridEhExportClass;
  i: integer;
  s: string;
  FileStream: TFileStream;
begin
{
Для экспорта лучше заменить заголовки колонок на имена полей. Заголовки колонок
экспортируются в первой строке. Имена полей более подходят для этой цели
}
  for i := 0 to XMLDataSet.FieldCount - 1 do
  begin
    DBGrid.Columns[i].Title.Caption := XMLDataSet.FieldDefs[i].Name;
  end;
  if not MultiSelect then
  begin
{
Для экспорта текущей записи нам приходится написать собственный кусок кода. Это
связано с особенностями работы использованного XMLDataSet. Он не совсем доработан
и экспорт отдельных строк с использованием метода DbGridEh получается плохо.
}
    FileStream := TFileStream.Create(ResultFileName, fmCreate);
    for i := 0 to XMLDataSet.FieldCount - 1 do
    begin
      s := AnsiQuotedStr(XMLDataSet.FieldDefs[i].Name, '"');
      if i <> XMLDataSet.FieldCount - 1 then
        s := s + ';' else s := s + #13#10;
```

```
      FileStream.Write(PChar(s)^, Length(s));
    end;
    for i := 0 to XMLDataSet.FieldCount - 1 do
    begin
      s := AnsiQuotedStr(XMLDataSet.Fields[i].AsString, '"');
      if i <> XMLDataSet.FieldCount - 1 then
        s := s + ';';
      FileStream.Write(PChar(s)^, Length(s));
    end;

    FileStream.Free;
  end
  else
  begin
{
Все строки экспортировать гораздо проще
}
    ExpClass := TDBGridEhExportAsCSV;
    SaveDBGridEhToExportFile(ExpClass, DBGrid,
      ResultFileName, MultiSelect);
  end;
  Close;
  ModalResult := mrOk;

end;

procedure TfrmGrid.FormShow(Sender: TObject);
begin
{
При показе формы восстанавливаем ее сохраненные размер и положение
}
  JvFormPlacement1.RestoreFormPlacement;
{
Открываем файл и настраиваем DbGrid
}
  InitGrid;
end;

initialization
end.
```

Далее создаем COM-сервер в виде dll примерно так, как мы делали для редактора XML. Текст модуля, в котором реализован единственный метод `Execute`, показан в листинге 16.15.

Листинг 16.15. Модуль u_XmlTable

```
unit u_XmlTable;
interface
uses
  ComObj, ActiveX, ruXmlTableSrv_TLB, StdVcl;
```

```
type
  TruXMLtableEditor = class(TAutoObject, IruXMLtableEditor)
  protected
    function Execute(const Dlg_Caption, Xml_FileName, Result_FileName,
      Storage_FileName, Ok_Caption, Cancel_Caption: WideString; Read_Only,
      Multi_Select: WordBool): OleVariant; safecall;
    { Protected declarations }
  end;

implementation
uses ComServ, frmXmlTableLite, Forms, Controls, Dialogs;

var
  Grid: TfrmGrid;

function TruXMLtableEditor.Execute(const Dlg_Caption, Xml_FileName,
  Result_FileName, Storage_FileName, Ok_Caption,
  Cancel_Caption: WideString; Read_Only,
  Multi_Select: WordBool): OleVariant;
begin
{
Реализация нашего единственного метода
}
  Grid := TfrmGrid.Create(Application);
  with Grid do begin
{
Устанавливаем переменные формы в соответствии с параметрами функции
}
    Caption := Dlg_Caption;
    XmlFileName := Xml_FileName;
    ResultFileName := Result_FileName;
    StorageFileName := Storage_FileName;
    XMLDataSet.ReadOnly := Read_Only;
    MultiSelect := Multi_Select;
    OkCaption:=Ok_Caption;
    CancelCaption:=Cancel_Caption;
{
Показываем форму для редактирования
}
    Result := (ShowModal = mrOk);
{
Результат функции, анализируемый программой-клиентом
}
    Free;
  end;
end;

initialization
  TAutoObjectFactory.Create(ComServer, TruXMLtableEditor, Class_ruXMLtableEditor,
    ciMultiInstance, tmApartment);
end.
```

И наконец, делаем тестовую программу-клиент для проверки работы COM-сервера (листинг 16.16).

Листинг 16.16. Модуль frmTest.pas

```
unit frmTest;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Mask, JvToolEdit, ruXmlTableSrv_TLB;

type
  TForm1 = class(TForm)
    JvFilenameEdit1: TJvFilenameEdit;
    Memo1: TMemo;
    Button1: TButton;
    CheckBox1: TCheckBox;
    CheckBox2: TCheckBox;
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
    ruXmlTable: IruXMLtableEditor;
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}
uses comobj;

procedure TForm1.FormCreate(Sender: TObject);
begin
{
При создании формы создаем COM-объект
}
  ruXmlTable := CreateComObject(CLASS_ruXMLtableEditor)
                                  as IruXMLtableEditor;
end;

procedure TForm1.Button1Click(Sender: TObject);
var
  XmlFName,
  OkCapt,StorageFname,  ResultFName: string;
  ResDial: OleVariant;
```

```
begin
  Memo1.Lines.Clear;
  XmlFName := JvFilenameEdit1.FileName;
  if ((XMLFname <> '') and FileExists(XMLFname)) then
  begin
    StorageFname := ChangeFileExt(XmlFname, '.ini');
    ResultFName := ChangeFileExt(XmlFname, '.csv');
    if CheckBox2.Checked then
    OkCapt := 'Выбрать все' else OkCapt := 'Выбрать строку';
    ResDial := ruXmlTable.Execute ('Таблица '+ ExtractFileName(XmlFname),
    XmlFName, ResultFName, StorageFname,OkCapt, 'Закрыть',CheckBox1.Checked,
CheckBox2.Checked
   );
    if ResDial then
    begin
      Memo1.Lines.LoadFromFile(ResultFName);
    end
    else
      ShowMessage('Отменено');
  end
  else
    ShowMessage('Не найден ' + XMLFname);
end;
end.
```

Запуск тестового проекта показал, что все работает (рис. 16.13).

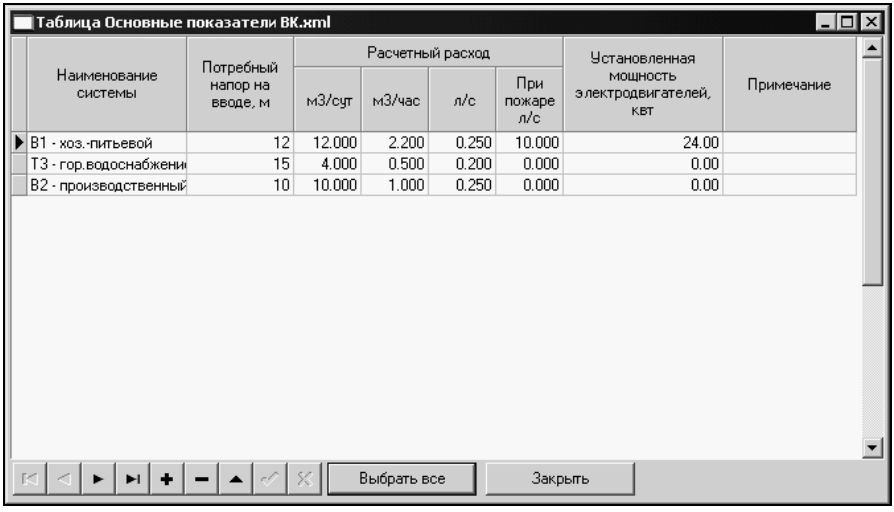


Рис. 16.13. Редактирование таблицы основных показателей ВК

Мы написали очень мало текста, но достигли приличного результата. Наша программа умеет делать очень много:

- отображать таблицу в привычной для пользователя форме (обратите внимание на многострочные заголовки, особенно на группу колонок под общей шапкой "Расчетный расход";
- колонки могут иметь мини-справочники, мы можем задавать их ширину и формат отображения данных;
- приложение "само по себе" сохраняет все изменения;
- одна или все строки экспортируются в простой текстовый формат, который легко может быть прочитан из LISP-программ.

Кроме того, наше приложение имеет большие возможности для расширения. Например, мы легко можем добавить строчку с суммой значений числовых колонок, сделать вывод таблицы на печать (все это благодаря замечательному компоненту Дмитрия Большакова). Отвлекаться на это мы пока не будем, т. к. это относится скорее к Delphi-технологиям, чем к САПР.

> Напоследок раскроем "ужасную тайну". Для того чтобы из нашего приложения получить доступ к свойствам полей объекта `XMLDataSet.xfields`, нам пришлось перенести объявление `xfields` в исходном тексте компонента из `private` в `public`. Конечно, более цивилизованным решением было бы создание собственного наследника, но тогда нам пришлось бы слишком много объяснять про компоненты. Мы решили просто исправить одну строчку.

LISP-функции для обработки таблиц

Итак, мы имеем приличный COM-сервер, который можно вызывать из LISP. Это уже хорошо. Попробуйте написать на LISP подобный диалог, да не один, а десятки. А мы сейчас получим единственную функцию, способную работать с любой таблицей. Но сначала продумаем общую стратегию работы с таблицами в AutoCAD.

Шапки таблиц у нас хранятся в файлах в каталоге %ruCADRootDir%\All Users\table\.

Таблица вставляется в виде блока и расчерчивается функцией `ru-table-draw-with-ask`. Размеры граф таблицы записаны в файле имя_таблицы.ini. Файл блока таблицы и INI-файл можно создать вручную, а можно и с помощью специальной программы, которую мы еще прокомментируем. Сейчас мы имеем около 70 таблиц, унаследованных от системы BestIA, переделывать их описания нет ни желания, ни времени. Мы просто дополним таблицу ее описанием в формате XML. Это будет точно такой же файл, какой мы разобрали на примере основных показателей ВК, только в нем не будет заполненных строк. А могут и быть, если в каких-то таблицах часто повторяется одинаковое содержание. Этот XML-файл будет являться прототипом для рабочих таблиц, заполняемых на конкретный объект. Рабочая таблица будет храниться в каталоге проекта под именем, заданным пользователем.

Схема работы с таблицей:

1. Выбор через XML-меню (листинг 16.17) таблицы.
2. Проверка наличия прототипа и, при его отсутствии, создание в этом же редакторе таблиц — эту работу делает пользователь с правами администратора ruCAD (рис. 16.14).

3. Задание имени рабочего файла.
4. При отсутствии рабочего файла копирование прототипа в рабочий файл.
5. Редактирование рабочего файла.
6. При завершении редактирования с экспортом данных запрос точки первой строки таблицы.
7. Определение ширины колонок из INI-файла.
8. Заполнение таблицы текстом.

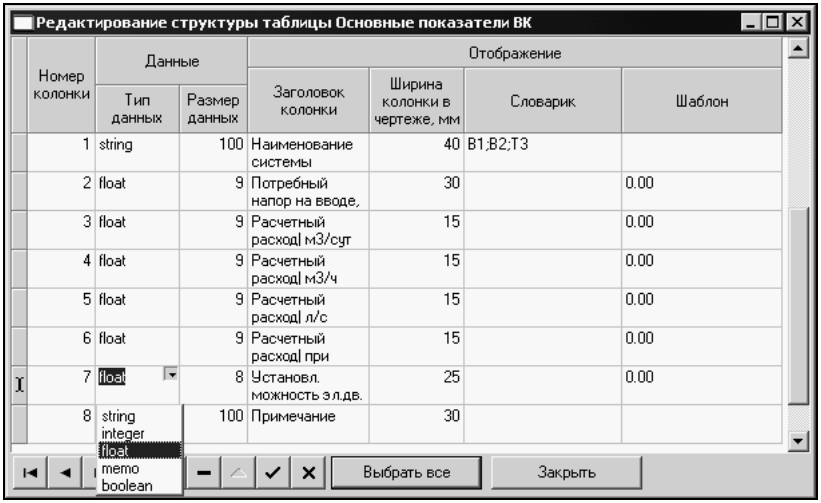


Рис. 16.14. Редактирование структуры таблицы

Листинг 16.17. Фрагмент XML-меню с вызовом программы

```
<?xml version='1.0' encoding='windows-1251'?>
<root name='Таблицы ВК' macro=''>
  <item name='Осн. показатели ВК' image='WK\TABVK1.GIF'
     comment='Рисует и заполняет таблицу с разграфкой'
     macro='(ru-table-xml-edit-and-draw-txt nil nil)'/>
</root>
```

Функция, реализующая работу с таблицей, приведена в листинге 16.18.

Листинг 16.18. Файл ru-table-xml-edit-and-draw-txt.lsp

```
(defun ru-table-xml-edit-and-draw-txt (block_table_name
 title_attrib_string is_ask_start_pnt / full_csv_file_name
 full_work_file_name ini_file_name lst_string_col lst_width_col
 pnt_left_first_col pnt_top_right _table-conv-csv-to-string-list
 _table-draw-all-strings)
```

```
;|
Параметры:
block_table_name - имя блока таблицы, без расширения
title_attrib_string - значение атрибута шапки или NIL, если нет
is_ask_start_pnt - надо ли запрашивать начало шапки таблицы
Пример:
(ru-table-xml-edit-and-draw-txt "Основные показатели ВК"  nil nil)
|;
;;;############ Вспомогательные функции ##############
;;; рисование текста таблицы по данным из CSV
  (defun _table-draw-all-strings (lst_string_col lst_width_col
pnt_left_first_col is_ask_start_pnt / pnt tmp ang shift)
;|
Заполнение таблицы текстом
Параметры:
lst_string_col - список списков для колонок
каждый подсписок - список строк по колонкам с шириной
(
("Текст колонки 1" "Текст колонки 2")
("Текст колонки 1" "Текст колонки 2")
)
lst_width_col - список ширин колонок в единицах рисунка
pnt_left_first_col - точка левого нижнего угла первой строки
is_ask_start_pnt - запрашивать ли новую точку начала таблицы
|;
;; чуть отодвигаем от угла
 (setq ang (ru-geom-go-left 0)
       shift (ru-conv-millimeter-in-paper-to-unit 3.0)
       pnt (polar pnt_left_first_col ang shift)
);_ end of setq
 (foreach lst_row lst_string_col
   (if is_ask_start_pnt
     (if (setq tmp (_ru-get-with-default
                     "Левый нижний угол следующей строки таблицы"
                     "Авто"'getpoint nil nil nil
                      );_ end of _ru-get-with-default
       );_ end of setq
       ;; вернет или указанную точку, или nil, при Enter
        (setq pnt (polar tmp ang shift))
    );_ end of if
  );_ end of if
  (setq pnt (ru-text-draw-in-table pnt lst_row lst_width_col))
);_ end of foreach
    pnt
);_ end of defun

  (defun _table-conv-csv-to-string-list (full_csv_file_name / result x)
;;; Преобразование csv в список для _table-draw-all-strings
```

```
;|
Формат csv:
Первая строка - имена полей колонок
Вторая и до конца - тексты колонок
"система";"напор";"м3_сутки";"м3_час";"л_секс";"пожарный";"квт";"прим"
"Т3 - гор.водоснабжение";"15";"4";"0.5";"0.2";"0";"0";""
|;
  (foreach string (cdr (ru-list-read-from-file full_csv_file_name))
   (setq result (cons
                (mapcar '(lambda (x) (ru-string-rem-all x (list "\"")))
                    (ru-string-to-list  string ";"))
                 result
              );_ end of cons
  );_ end of setq
 );_ end of foreach
 (vl-file-delete full_csv_file_name)
 (reverse result)
);_ end of defun
;;;******************** Главная функция **********************
 (setq ini_file_name
       (ru-file-set-ext (ru-file-table block_table_name) ".ini")
        lst_width_col (ru-table-make-lst-width block_table_name)
)
;;;
 (if (ru-yes (strcat
        "Рисование таблицы."
        "\nЕсли нужно нарисовать таблицу, "
        " при запросе левого нижнего угла"
        "\nвыберите опцию Draw\n Продолжить?"
     );_ end of strcat
   );_ end of ru-yes
    (progn
       (setq pnt_left_first_col (ru-get-point-or-exit
                "Левый нижний угол первой строки таблицы" "Draw")
      );_ end of setq
       (cond
         ((= pnt_left_first_col "Draw")
        ;; Рисуем таблицу с разграфкой и получаем
        ;; точку правого верхнего угла для
        ;; возможного продолжения
          (setq pnt_top_right (ru-table-draw-with-ask block_table_name
                                    title_attrib_string T
                                );_ end of ru-table-draw-with-ask
                pnt_left_first_col (polar (polar (polar pnt_top_right
         (ru-geom-go-back 0) (ru-conv-millimeter-in-paper-to-unit
         (apply '+ lst_width_col))) (ru-geom-go-right 0)
         (ru-conv-millimeter-in-paper-to-unit (atof
         (ru-ini-read ini_file_name "Table" "header_height" "0"))))
         (ru-geom-go-right 0) (ru-normal-table-row-height))))
      );_ end of cond
```

```
;; Запрос имени файла
 (if (ru-yes (strcat "Редактирование таблицы."
                "\nЗадайте имя рабочего файла таблицы"
                "\nВ рабочем файле данные будут сохраняться"
                "\n Продолжить?"
              );_ end of strcat
   );_ end of ru-yes
  (if
    (setq full_work_file_name
           (ru-dlg-file-select-or-new
             "Выбери или создай рабочий файл таблицы"
             (nth 0 (dos_splitpath (getvar "DWGPREFIX")))
             (getvar "DWGPREFIX") "XML-таблицы ruCAD" "*.ruxt"
           );_ end of ru-dlg-file-select-or-new
    );_ end of setq
      (if (setq full_csv_file_name
                 (ru-table-xml-edit-work-file
                   block_table_name
                   full_work_file_name
                 );_ end of ru-table-xml-edit-work-file
          );_ end of setq
        (if (setq lst_string_col
                   (_table-conv-csv-to-string-list
                     full_csv_file_name
                   );_ end of
            );_ end of setq
         ;; Заполняем таблицу
          (_table-draw-all-strings lst_string_col lst_width_col
            pnt_left_first_col is_ask_start_pnt
          );_ end of _table-draw-all-strings
          (princ "\nОШИБКА: _table-conv-csv-to-string-list")
        );_ end of if
         (princ "\nОтказ от экспорта из рабочего файла таблицы")
      );_ end of if
       (princ "\nОтказ от выбора рабочего файла таблицы")
   );_ end of if
    (princ "\nОтказ от редактирования рабочего файла таблицы")
 );_ end of if
  (princ "\nОтказ от рисования")
);_ end of progn
);_ end of if
);_ end of defun
```

Примерный вид во время работы функции `ru-table-xml-edit-work-file` показан на рис. 16.13.

Примерный вид таблицы, нарисованной и заполненной в DWG-файле см. на рис. 16.15.

Функция, рисующая таблицу с запросом расположения шапки, приведена в листинге 16.19.

ОСНОВНЫЕ ПОКАЗАТЕЛИ ПО ЧЕРТЕЖАМ ВОДОПРОВОДА И КАНАЛИЗАЦИИ							
Наименование системы	Потребный напор на вводе, м.вод.ст.	Расчетный расход				Установл. мощность эл. двигателей, кВт	Примечание
		м3/сут	м3/ч	л/с	при пожаре л/с		
В1 - хоз.-питьевой	20.00	10.00	1.00	0.15	10.00	15.00	
Т3 -	15.00	5.00	1.00	0.20	0.00	0.00	
гор.водоснабжение							
В2 -	15.00	30.00	1.50	0.25	0.00	0.00	
производственный							

Рис. 16.15. Заполненная таблица в DWG-файле

Листинг 16.19. Функция `ru-table-draw-with-ask`

```
(defun ru-table-draw-with-ask (block_table_name title_attrib_string
                               is_vertical / block_ins_pnt)
;|Параметры:
block_table_name - имя блока таблицы, без расширения
title_attrib_string - значение атрибута шапки или NIL, если нет
is_vertical  - T, если таблица вертикальная, Nil - если горизонтальная
 Возвращает точку правого верхнего угла для возможного продолжения или nil
Особенности:
1. title_attrib_string передается, если ДЕЙСТВИТЕЛЬНО есть ОДИН и ТОЛЬКО один
атрибут. Имя его неизвестно. Обычно это таблица с постоянной шапкой,
но с переменным заголовком: ВЕДОМОСТЬ ССЫЛОЧНЫХ И ПРИЛАГАЕМЫХ ДОКУМЕНТОВ,
ВЕДОМОСТЬ ОСНОВНЫХ КОМПЛЕКТОВ, ВЕДОМОСТЬ НОРМАТИВНЫХ И ИСХОДНЫХ ДОКУМЕНТОВ
Контроль за применением этого параметра возлагается на программиста
2. Горизонтальные таблицы не подлежат разграфке. Обычно это заготовки профилей.
Вертикальные таблицы разграфляются по запросу
Эта функция применяется и в программах, где производится только рисование таблиц,
без их заполнения
Примеры:
 (ru-table-draw-with-ask "Основные показатели ВК"  nil T)
 (ru-table-draw-with-ask "VedSer" "ВЕДОМОСТЬ НЕНОРМАТИВНЫХ ВЫРАЖЕНИЙ" T)
(ru-table-draw-with-ask "PROGS_N" nil nil)
|;
  (if (setq block_ins_pnt (ru-get-point-or-exit (strcat "Левый "
                          (if is_vertical "верхний" "нижний") " угол формы"
              ) nil))
    (ru-table-grid  block_table_name title_attrib_string block_ins_pnt
      (ru-get-point-required
        (strcat
          (if is_vertical "Левый " "Правый ")
          " нижний угол формы"
       );_ end of strcat
        block_ins_pnt
     );_ end of ru-get-point-required
```

```
      is_vertical
      is_vertical
   );_ end of ru-table-grid
    nil
 );_ end of if
);_ end of defun
```

Функция, выполняющая рисование таблицы с заданными параметрами, приведена в листинге 16.20.

Листинг 16.20. Функция `ru-table-grid`

```
(defun ru-table-grid (block_file_name title_attrib_string block_ins_pnt
     end_pnt is_vertical is_ask_grid_draw / ini_file_name column_width
    header_height lst_column_width pnt1 pnt2 result row_count table_angle
    table_block table_height)
;|
Собственно отрисовка таблицы. Параметры block_file_name и title_attrib_string
аналогичны предыдущей функции
 block_ins_pnt   - угол шапки формы (левый верхний для вертикальных
 таблиц и левый нижний для горизонтальных)
 end_pnt  - угол юбки формы (левый нижний для вертикальных и правый нижний для
горизонтальных)
 is_vertical  - T, если таблица вертикальная, nil - если горизонтальная
 is_ask_grid_draw - запрашивать разграфку (действует только для вертикальных)
 Возвращает точку правого верхнего угла  формы для возможного продолжения или NIL
|;
 (setq ini_file_name
        (ru-file-set-ext (ru-file-table block_file_name) ".ini")
 );_ end of setq
  (if (findfile ini_file_name)
      (progn
        (setq lst_column_width  (ru-table-make-lst-width block_file_name)
              table_block      (ru-file-table
                                 (ru-ini-read
                                   ini_file_name
                                   "Table"
                                   "header_block"
                                   ""
                                  );_ end of ru-ini-read
                                );_ end of ru-file-table
              header_height
                                 (atof (ru-ini-read
                                         ini_file_name
                                         "Table"
                                         "header_height"
                                         "0"
                                       );_ end of ru-ini-read
                                );_ end of atof
      );_ end of setq
```

```
        (if (/= table_block "")
          (progn
            (ru-var-clear-osnap)
            (setvar "ATTDIA"
                    (ru-conv-bool-to-int (null title_attrib_string))
            );_ end of setvar
            (command
              "_.INSERT"
              table_block
              "_Scale"
              (ru-conv-millimeter-to-unit (ru-scale-current-space))
              "_ROTATE"
              "0"
              block_ins_pnt
            );_ end of command
            (if title_attrib_string
              (command title_attrib_string)
            );_ end of if
            (ru-var-restore-osnap)
          );_ end of progn
        );_ end of if
        (if is_vertical
          (setq table_angle (ru-geom-go-right 0))
          (setq table_angle 0)
        );_ end of if
        (setq pnt1 (polar block_ins_pnt table_angle
                    (ru-conv-millimeter-in-paper-to-unit header_height)
                  );_ end of polar
          row_count       (ru-match-ceiling
                            (/ (distance pnt1 end_pnt)
                               (ru-normal-table-row-height)
                            );_ end of /
                          );_ end of ru-match-ceiling
          table_height  (* row_count (ru-normal-table-row-height))
          ;; уточненная высота
          pnt2            (polar pnt1 table_angle table_height)
          ;; уточненный угол
          end_pnt pnt2
        );_ end of setq
        ;; край формы
        (ru-line-add pnt1 pnt2 0;|(ru-lw-normal)|; nil)
(foreach column lst_column_width
          (setq column_width column
                pnt1 (polar pnt1 (ru-geom-go-left table_angle)
                           (ru-conv-millimeter-in-paper-to-unit olumn_width)
                     );_ end of polar
                pnt2 (polar pnt2 (ru-geom-go-left table_angle)
                           (ru-conv-millimeter-in-paper-to-unit column_width)
                     );_ end of polar
          );_ end of setq
```

```
      ;; Вертикальные линии
       (ru-line-add pnt1 pnt2 0 nil)
       );_ end of foreach
       ;; Низ формы
        (ru-line-add end_pnt pnt2 0 nil)
        (if is_vertical
          (setq
            result
             (polar
               pnt2
               (ru-geom-go-back table_angle)
               (+ table_height
                  (ru-conv-millimeter-in-paper-to-unit header_height)
               );_ end of +
             );_ end of polar
         );_ end of setq
          (setq result pnt2)
       );_ end of if
        (if (and  is_vertical
                  is_ask_grid_draw
                 (ru-yes "Стандартно разграфить форму")
           );_ end of and
          (repeat (- row_count 1)
            (setq pnt2 (polar pnt2 (ru-geom-go-back table_angle)
                                   (ru-normal-table-row-height)
                       );_ end of polar
                  end_pnt (polar end_pnt (ru-geom-go-back table_angle)
                                         (ru-normal-table-row-height)
                          );_ end of polar
           );_ end of setq
            (ru-line-add end_pnt pnt2 0  nil)
         );_ end of repeat
       );_ end of if
     );_ end of progn
    (ru-msg-alert (strcat "Не найден файл " ini_file_name))
 );_ end of if
  result
);_ end of defun
```

В приведенных листингах вы встретились со множеством вспомогательных функций ruCAD. Надеемся, что их назначение понятно из названий, а тексты вы найдете в других главах или на прилагаемом компакт-диске. Мы затратили достаточно времени и книжных площадей, но зато получили универсальные средства для рисования и заполнения любых таблиц для любых разделов проекта. Остальное "программирование" таблиц сводится к заполнению элементов XML-меню.

ГЛАВА 17

Разработка диалогового окна выбора файлов

В любой прикладной системе находится множество применений для диалоговых окон, в которых производится выбор файлов, но разработчики некоторых, даже очень солидных систем, относятся к этому безответственно. Раз можно использовать стандартный вид диалоговых окон Windows, то и будем его использовать. В результате мизерной экономии страдают миллионы потребителей. Вам не приходилось проклинать авторов продукта, в котором нужно производить выбор нескольких десятков файлов, делая это в миниатюрном окошечке, размеры которого нельзя изменить? Да еще используя комбинацию клавиши <Ctrl> и кнопки мыши для выделения имен? Не приходилось, случайно отпустив клавишу, терять набор и начинать все сначала? Действительно не приходилось? Тогда вы счастливый в своем неведении человек, у которого все еще впереди.

Не зря множество программистов занимаются разработкой библиотек компонентов для создания файловых диалоговых окон, и есть даже фирмы, у которых это основная продукция.

Нам тоже понадобятся диалоговые окна открытия файлов, да не простых, а файлов рисунков системы AutoCAD, имеющих сложный формат. Разработкой таких диалоговых окон мы и займемся в этой главе.

Чем нас не устраивают стандартные диалоговые окна

Со стандартными диалоговыми окнами системы AutoCAD нам приходится сталкиваться в двух случаях — в команде OPEN и при использовании функции `getfiled`. В первом случае нам деваться некуда и мы можем только одобрять или критиковать решения фирмы Autodesk. Некоторый прогресс (если сравнивать версии AutoCAD R12 и AutoCAD 2004) все-таки есть. Правда, кое-что сделано по известному принципу "хотели как лучше". Например, модная панель быстрого доступа "в стиле Outlook", миниатюры предварительного просмотра прямо в списке файлов.

Может быть "где-то на Западе" пользователи и рукоплещут этим "приколам" программистов, но мы подозреваем, что наши "тетки" не будут щелкать по пиктограмме Buzzsaw ("непереводимая игра слов"), не будут выкладывать рисунки на Desktop (Ра-

бочий стол), а "продвинутые" пользователи, имеющие доступ в Интернет, не будут помещать рисунки в Favorites (Избранное) — эта папка и так ломится от тысяч ссылок на интернет-ресурсы. При необходимости пользователь может выбрать любую виртуальную папку из раскрывающегося списка, а драгоценную площадь основного окна можно использовать с большей пользой.

Примерно пятая часть площади диалогового окна под миниатюрой предварительного просмотра совершенно не используется, хотя именно сюда можно было бы поместить основные реквизиты выделенного файла, сейчас надежно спрятанные от пользователя. Как же "тетка" додумается, что для просмотра свойств файла нужно вызвать контекстное меню? А потом, не испугавшись появившегося списка, выбрать в самом низу пункт свойства и ознакомиться с комментариями к файлу? Ну почему нельзя вывести прямо в основное диалоговое окно не только миниатюру предварительного просмотра, но и основные дополнительные атрибуты, хотя бы с вкладки **Summary** (Документ)?

Диалоговые окна, выводимые функцией `getfiled`, еще хуже. В одних режимах они позволяют изменить размер окна для просмотра больших списков файлов, в других — размер диалога остается неизменным, но в него еще вписываются миниатюры предварительного просмотра (включая изображение папки с вписанными в нее миниатюрами файлов). По поводу остальных мелочей просто не хватает таких слов, которые не вычеркнет редактор этой книги, поэтому мы просто сделаем собственные диалоговые окна, работающие так, как нам нужно.

Давайте сформулируем, что же мы желаем иметь в нашем диалоговом окне.

Ограничение навигации

Во многих случаях необходимо ограничить навигацию пользователя по дереву папок определенным каталогом. Ниже ограничивающего каталога пользователь может просматривать содержимое папок, а подниматься выше и переходить на другие диски — не должен. Ограничивающий каталог должен задаваться в контексте вызывающей программы.

Очевидно, должна быть возможность задания стартового каталога, с которого начинается просмотр. Очень раздражает, когда в программах выбор файла начинается с какого-либо постоянного места, например с папки Мои документы. Если мы предусмотрели персональные рабочие каталоги для пользователей, то, например, выбор файла для загрузки в AutoCAD должен начинаться именно с этого каталога.

Должна быть возможность задания маски отображаемых файлов. В случае создания нового файла, если пользователь не указал явно расширение, заданное расширение должно присваиваться автоматически.

Предварительный просмотр DWG-файлов

Предварительный просмотр DWG-файлов безусловно нужен, но мы сделаем просмотр только выделенного файла, а не будем демонстрировать программистские трюки по внедрению миниатюры предварительного просмотра (Thumbnail) в список файлов. Для просмотра растровой картинки, зашитой в DWG-файл, мы создали свой компонент, ruDwgPreview (см. *главу 21*, листинг 21.9).

Комментирование файлов и папок

Создание комментариев к файлам и папкам имеет чрезвычайно важное значение. Особенно актуально это было при работе в DOS, когда пользователи вынуждены были давать файлам невразумительные короткие имена, да еще непременно латинскими символами. Хотя после появления возможности присваивать файлам длинные имена острота проблемы несколько снизилась, наличие краткого описания папки или документа на родном языке пользователя никогда не повредит. Многие программы позволяют комментировать файлы. Мы уже упоминали о комментировании файлов в файловом менеджере Total Commander, позволяющем "на лету" комментировать файлы и папки нажатием комбинации клавиш <Ctrl>+<Z> и имеющем специальный режим просмотра имен файлов с комментариями.

В нашей старой системе BestIA мы принудительно *заставляли* раздраженного пользователя при создании файла давать ему комментарий и *рекомендовали* все-таки использовать короткие латинские имена. Десять лет работы показали правильность этого решения. В нескольких "подшефных" организациях после "лечения" файловой системы очередной версией утилиты Norton Disk Doctor терялись длинные русские имена файлов, и только наличие файлов dirinfo.ini *(см. главу 3)* позволяло разобраться с бесценными архивами документов. В других организациях наличие комментариев, сделанных еще DOS-версиями системы, позволяет удобно использовать старые файлы, хотя уже никто не помнит, что за файл *12\086\123.dwg* (доступен комментарий, что это *"Типовые секции\Серия 86-06\Планы первого, типового и верхнего этажей"*).

Полезной оказалась возможность изменения комментариев, без изменения физических имен файлов, и такую возможность мы сохраним. Пользователь с определенными привилегиями будет иметь возможность в процессе выбора файла изменить комментарии к другим файлам и папкам.

Поддержка комментариев к любым документам настолько необходима, что фирма Microsoft разработала формат общего применения для потока аннотирующей информации о документах в виде файла хранения структурированных данных. Когда-то такая информация хранилась, например, в документах Microsoft Word в собственном формате, доступном только через само приложение Word. Теперь аннотирующая информация может храниться в виде набора свойств *OLE* (Ole Property Set) и доступ к ней открыт из любого другого приложения, поддерживающего интерфейсы `IPropertySetStorage` и `IPropertyStorage`. К сожалению, эти замечательные новинки появились слишком поздно, но, к счастью, фирма Autodesk, начиная с AutoCAD 2000, стала их использовать и теперь мы имеем возможность программного доступа ко всем свойствам файла, отображаемым и изменяемым командой системы AutoCAD DWGPROPS (СВОЙСТВАРИС), без запуска самой системы AutoCAD, и даже без ее установки на компьютер. Далее мы узнаем, как это делается.

Логика получения аннотирующей информации

Аннотирующая информация может храниться в разных местах:

- внутри самого документа;
- в расширенных атрибутах файловой системы NTFS;
- в различных внешних файлах, таких как descript.ion или выдуманные нами dirinfo.ini.

Информация, находящаяся внутри DWG, естественно, является самой актуальной. Сведения, получаемые из файловой системы, теоретически, должны быть такими же. Но всякое бывает. Перенесите на дискете отлично аннотированный в Windows 2000 или Windows XP документ на другой компьютер и попробуйте там просмотреть его свойства.

Сведения, записанные в INI-файлах, самые ненадежные, т. к. их очень легко изменять, но, с другой стороны, это и хорошо. Комментарии из dirinfo.ini мы можем использовать для быстрого аннотирования документов. Если эти комментарии потеряются, это не будет иметь особых последствий, но они же очень помогут во многих случаях.

Просмотр и редактирование свойств DWG-файлов без AutoCAD

Все свойства DWG-файла просматриваются и редактируются внутри системы AutoCAD с помощью команды DWGPROPS (СВОЙСТВАРИС). Мы уже договорились о стандарте использования свойств *(см. главу 3)*. К команде DWGPROPS (СВОЙСТВАРИС) мы можем прицепить реактор, обрабатывающий изменения, внесенные во время исполнения диалога.

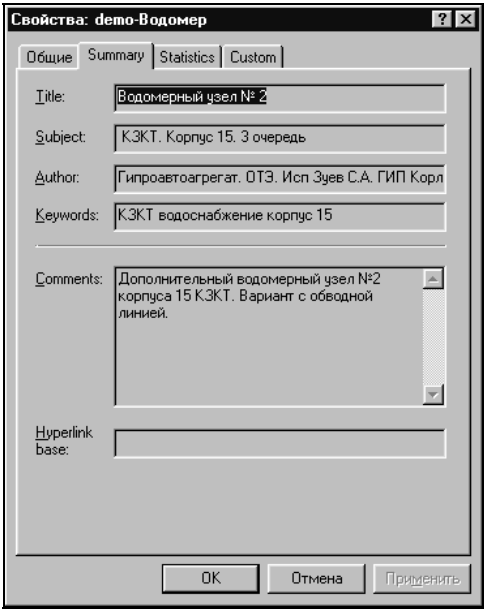


Рис. 17.1. Просмотр свойств DWG-файла на компьютере с AutoCAD

Если на компьютере установлена система AutoCAD, то свойства рисунка просматриваются и в диалоговом окне свойств, вызываемом из контекстного меню любого навигатора файловой системы (рис. 17.1). Делается это средствами Windows. Причем посмотреть-то можно, а извлечь информацию (например, для использования в системе документооборота) нельзя. Если на компьютере система AutoCAD не установ-

лена, то свойства рисунка можно увидеть только при использовании файловой системы NTFS, т. е. в Windows NT, Windows 2000 или Windows XP.

Разработка диалогового окна

Сначала рассмотрим диалоговое окно (рис. 17.2) с точки зрения пользователя, познакомимся с его возможностями, а потом разберем, как это делается с точки зрения программиста.

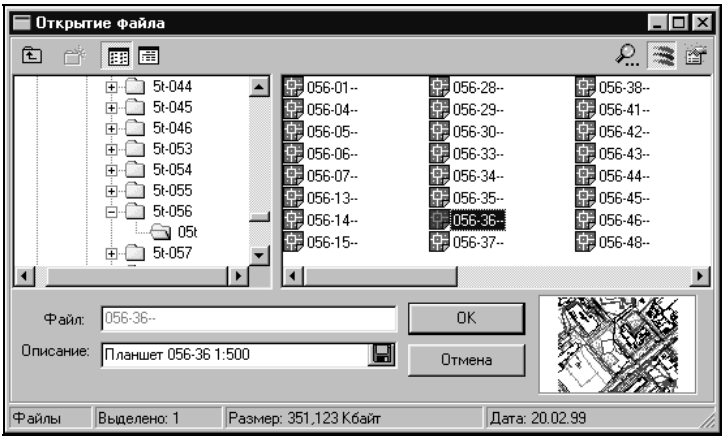


Рис. 17.2. Диалоговое окно открытия файла

Диалоговое окно имеет заголовок, текст которого задается в виде параметра. Судя по кнопкам на заголовке окна, оно может быть развернуто во весь экран (положение и размер диалогового окна запоминаются для данного пользователя). На панели инструментов имеются кнопки для перехода на вышележащий уровень, создания новой папки, переключения отображения перечня файлов между форматами список—таблица, поиска файлов (рис. 17.3), включения-отключения предварительного просмотра и вызова диалогового окна просмотра свойств файла (рис. 17.1).

В левой части диалогового окна отображается дерево папок, в правой — список файлов, находящихся в выделенной папке. В нижней части диалогового окна имеются поля редактирования имени файла и комментарии к выделенному файлу или папке. Комментарий запоминается при щелчке по кнопке с изображением дискеты. Для выделенного файла отображается миниатюра предварительного просмотра.

При щелчке по кнопке поиска файлов вызывается диалоговое окно (рис. 17.3), в котором можно задать условия поиска, просмотреть свойства, комментарии и миниатюры найденных файлов и, щелкнув по кнопке **OK**, сразу получить результат выбора, не возвращаясь к основному диалоговому окну.

Свойства диалогового окна устанавливаются аргументами вызывающей функции. Программист может задать:

- заголовок диалогового окна;
- стартовый каталог, в котором начинается выбор;

- ограничивающий каталог, выше которого пользователь не может подняться (если такой каталог не задан, доступна навигация по всем папкам, включая виртуальные и сетевое окружение);
- маску отображаемых файлов;
- режим выбора одного или нескольких файлов;
- разрешение создавать новый файл;
- разрешение на редактирование комментариев.

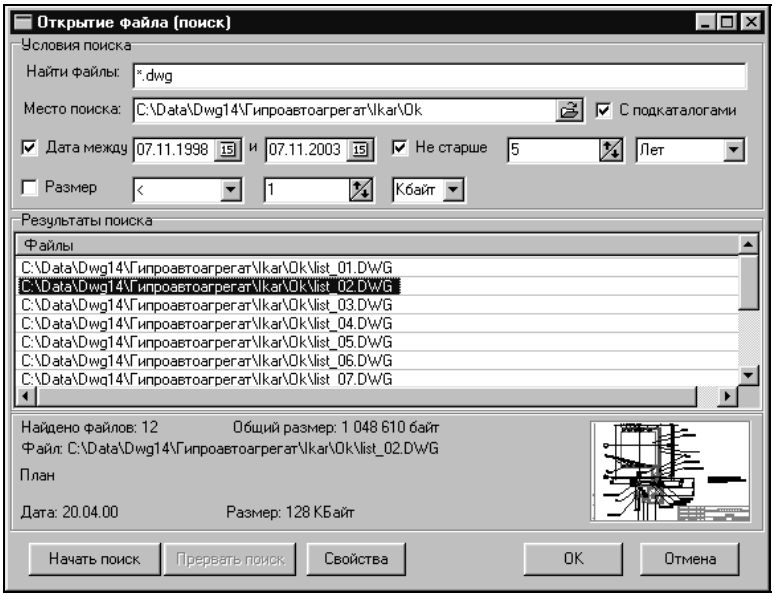


Рис. 17.3. Диалоговое окно поиска файлов

Для реализации всех указанных возможностей необходимы специальные нестандартные компоненты. Таких компонентов имеется много, но с учетом нашего стремления обойтись только бесплатными компонентами с доступными исходными текстами мы остановились на библиотеке ShellShock известной фирмы TurboPower[1]. Компоненты этой библиотеки `TstShellTreeView` и `TstShellListView` имеют весь необходимый нам набор свойств и методов, позволяющий обойтись минимальными собственными дополнениями[2].

---

[1] Вообще-то до недавнего времени это были платные продукты фирмы, известной своими библиотеками Turbo Professional, Object Professional, Orpheus, SysTools и многими другими. Но в начале 2003 года фирма TurboPower неожиданно объявила о прекращении коммерческого распространения библиотек и практически вся ее продукция вместе с исходными текстами стала доступна на известном хранилище проектов с открытым кодом **sourceforge.net**.

[2] Впоследствии мы будем использовать еще более удобную библиотеку компонентов VirtualShellTools (**www.mustangpeak.net**), но для понимания технологии это не принципиально.

Мы не будем тратить место на исходный текст формы, а более подробно прокомментируем текст модуля (листинг 17.1). Не имеющие принципиального значения фрагменты исходного текста пропущены, а участки кода, на которые нужно обратить внимание, выделены полужирным шрифтом.

Листинг 17.1. Файл frmRuShellFileDlg.pas

```
unit frmRuShellFileDlg;

interface

uses
  Windows, Messages, SysUtils, {Variants,} Classes,  Controls, Forms,
  Dialogs, Buttons, ruDwgPreview, Mask, JvToolEdit, JvPlacemnt,
  StShlCtl, JvStatusBar, ImgList, ComCtrls, ExtCtrls, StdCtrls;

type
  TformFileDlg = class(TForm)
    ShellList: TStShellListView;
    ShellTree: TStShellTreeView;
    LabelFile: TLabel;
    EditFileName: TEdit;
    LabelDescription: TLabel;
    EditDescription: TJvComboEdit;
    PanelPreview: TPanel;
    DWGpreview: TruDwgPreview;
    JvFormPlacement1: TJvFormPlacement;
    StatusBar: TJvStatusBar;
    btnPreview: TSpeedButton;
    btnSearch: TSpeedButton;
    btnMoveUp: TSpeedButton;
    btnCreateNewFolder: TSpeedButton;
    btnList: TSpeedButton;
    btnDetails: TSpeedButton;
    btnCancel: TButton;
    btnOk: TButton;
    btnProp: TSpeedButton;
    procedure FormCreate(Sender: TObject);
    procedure FormShow(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure FormDestroy(Sender: TObject);
    procedure ShellTreeFolderSelected(Sender: TObject;
      Folder: TStShellFolder);
    procedure ShellListFilterItem(Sender: TObject;
       ShellItem: TStShellItem; var Accept: Boolean);
    procedure ShellListChange(Sender: TObject; Item: TListItem;
      Change: TItemChange);
    procedure ShellListSelectItem(Sender: TObject; Item: TListItem;
      Selected: Boolean);
    procedure ShellListEditing(Sender: TObject; Item: TListItem;
      var AllowEdit: Boolean);
```

```
    procedure ShellListEdited(Sender: TObject; Item: TListItem;
      var S: string);
    function NewFileNameValid(FileName: string): boolean;
    procedure EditFileNameChange(Sender: TObject);
    procedure EditDescriptionChange(Sender: TObject);
    procedure EditDescriptionButtonClick(Sender: TObject);
    procedure btnListClick(Sender: TObject);
    procedure btnDetailsClick(Sender: TObject);
    procedure btnMoveUpClick(Sender: TObject);
    procedure btnCreateNewFolderClick(Sender: TObject);
    procedure btnPropClick(Sender: TObject);
    procedure btnPreviewClick(Sender: TObject);
    procedure btnSearchClick(Sender: TObject);
    procedure btnCancelClick(Sender: TObject);
    procedure btnOkClick(Sender: TObject);

  private
    { Private declarations }
      StorageIniFile, OldName, OldComment, DefaultExt: string;
      ActiveFolder:boolean;
  public
    { Public declarations }
{ Переменные, доступные из модуля реализации
}
    DlgCaption, // Заголовок окна
      RootDir, // Ограничивающий каталог
      StartDir, // Начальный каталог
      DefaultFileName, // Имя нового файла по умолчанию
      FilterMask: string; // Маска файлов
      MultiSelect, // Множественный выбор файлов
      CanEdit, //Редактирование имен и папок
    CanCreateNewFile: boolean; // Можно ли создавать новый файл
    StringListNames: TStringList; // Список выбранных файлов
  end;

var
  formFileDlg: TformFileDlg;

implementation
uses
  ruDwgInfo, ruFileUtils, ruUtils, JvFileUtil,  Masks, ssBase, JclShell,
  ComObj, ruFileSearchSvr_TLB;

{$R *.dfm}
procedure TformFileDlg.FormCreate(Sender: TObject);
begin
  DlgCaption := 'Проверка';
  RootDir := '';
  StartDir := '';
  DefaultFileName := 'Новый файл';
  FilterMask := '*.dwg';
  MultiSelect := False;
```

```
  CanEdit := True;
  CanCreateNewFile := True;

  StorageIniFile := IncludeTrailingBackslash(ruGet_LocalAppDataDir) +
    'FileDialog\FormStorage.ini';
   ShellList.OpenDialogMode :=False;
end;

procedure TformFileDlg.FormShow(Sender: TObject);
begin
  JvFormPlacement1.IniFileName := StorageIniFile;
  JvFormPlacement1.RestoreFormPlacement;
  Caption := DlgCaption;
  if RootDir <> '' then begin
    ShellTree.RootFolder := RootDir;
    ShellTree.SpecialRootFolder := sfNone;
  end;
  ShellTree.StartInFolder := StartDir;
  ShellList.FileFilter := FilterMask;
  if MultiSelect then  CanCreateNewFile := False;
  if (not CanCreateNewFile) then DefaultFileName := '';
  EditFileName.Text := DefaultFileName;
  ShellList.MultiSelect := MultiSelect;
  ShellList.ReadOnly := (not CanEdit);
  ShellTree.ReadOnly := (not CanEdit);
  DefaultExt := ruFileDefaultExt(FilterMask);
  EditFileName.Enabled := CanCreateNewFile;
  EditDescription.Enabled := CanEdit or CanCreateNewFile;
  btnOk.Enabled := False;
  btnCreateNewFolder.Enabled := CanEdit;
  StatusBar.Panels[0].Text:='';
  StatusBar.Panels[1].Text:='';
// Начало просмотра с заданного каталога
  ShellTree.SelectFolder(StartDir);
end;

procedure TformFileDlg.FormClose(Sender: TObject;
  var Action: TCloseAction);
begin
  JvFormPlacement1.SaveFormPlacement;
end;

procedure TformFileDlg.FormDestroy(Sender: TObject);
begin
  if Assigned(StringListNames) then
    StringListNames.Free;
end;

procedure TformFileDlg.ShellTreeFolderSelected(Sender: TObject;
  Folder: TStShellFolder);
{
Обработка события выбора папки в дереве
}
```

```
begin
  ActiveFolder:=True;
  DWGpreview.Visible := false;
  EditDescription.Text := '';
  EditDescription.Hint := 'Комментарий к папке';
  EditFileName.Text := '';
  EditFileName.Text := Folder.DisplayName;
// Показываем комментарий к папке
  EditDescription.Text :=
    ruReadDirinfoFolderDescription(Folder.Path);
  EditFileName.Hint := EditFileName.Text;
  btnOk.Enabled := False;
// Разрешение на редактирование комментария
  EditDescription.Button.Enabled := CanEdit and
      (EditDescription.Text <> '');
  btnCreateNewFolder.Enabled := True;
  StatusBar.Panels[0].Text:='Папка';
  StatusBar.Panels[1].Text:='';
  StatusBar.Panels[2].Text:='';
  StatusBar.Panels[3].Text:='';
end;

procedure TformFileDlg.ShellListFilterItem(Sender: TObject;
  ShellItem: TStShellItem; var Accept: Boolean);
{
Установка фильтра для показа файлов
}
begin
  Accept := ShellItem.IsFile and
             MatchesMask(ShellItem.Path, FilterMask);
end;

procedure TformFileDlg.ShellListChange(Sender: TObject; Item: TListItem;
  Change: TItemChange);
begin
  StatusBar.Panels[1].Text:='Выделено: ' +
    IntToStr(ShellList.SelectedItems.Count);
end;

procedure TformFileDlg.ShellListSelectItem(Sender: TObject;
  Item: TListItem; Selected: Boolean);
{
Обработка события изменения фокуса элемента в списке файлов
}
var
  SelectedName, SelectedPath, Ext: string;
begin
  ActiveFolder:=False;
  if (DefaultFileName <> '') and CanCreateNewFile then
    EditFileName.Text := DefaultFileName
  else
    EditFileName.Text := '';
```

```
  DefaultFileName := '';
  EditFileName.Hint := EditFileName.Text;
  EditDescription.Text := '';
  EditDescription.Hint := 'Комментарий к файлу';
  EditFileName.Hint := 'Редактор имени создаваемого файла';
  StatusBar.Panels[1].Text:='';
  StatusBar.Panels[2].Text:='';
  StatusBar.Panels[3].Text:='';
  DWGpreview.Visible := false;
  btnCreateNewFolder.Enabled := False;
  if Selected then
  begin
    btnOk.Enabled := True;
    btnProp.Enabled := ShellList.SelectedItem.HasPropSheet;
    SelectedName := ShellList.SelectedItem.DisplayName;
    SelectedPath := ShellList.SelectedItem.Path;
    EditFileName.Text := SelectedName;
    EditFileName.Hint := SelectedPath;
  if FileExists(SelectedPath) then
    begin
    StatusBar.Panels[2].Text := 'Размер: '+
             CommaizeL(ShellList.SelectedItem.Size) +' Кбайт';
    StatusBar.Panels[3].Text := 'Дата: '+
              DateToStr(ShellList.SelectedItem.Date);
     Ext := AnsiLowerCase(ExtractFileExt(SelectedPath));
//===== Комментарий к файлу ============================
     EditDescription.Text := ruReadDirinfoFileDescription(SelectedPath);
//===== Просмотр DWG ===================================
      if (Ext = '.dwg') or (Ext = '.dwt') then
      begin
        btnPreview.Visible:=True;
        try
          DWGpreview.FileName := SelectedPath;
        except
        end;
          DWGpreview.Visible := btnPreview.Down;
      end else begin
        btnPreview.Visible:=False;
        DWGpreview.Visible := False;
      end;
      EditDescription.Hint := EditDescription.Text;
      EditDescription.Button.Enabled := CanEdit and
                      (EditDescription.Text <>'');
    end;
  end;
  StatusBar.Panels[0].Text:='Файлы';
  StatusBar.Panels[1].Text:='Выделено: ' +
    IntToStr(ShellList.SelectedItems.Count);
end;
```

```
procedure TformFileDlg.ShellListEditing(Sender: TObject; Item: TListItem;
  var AllowEdit: Boolean);
begin
{
  Возникает в начале переименования файла в ShellList
  Надо запомнить прежнее имя и комментарий
  После переименования нужно переименовать INI-файл и заменить информацию в DIRINFO
}
  OldName := Item.Caption;
  OldComment := EditDescription.Text;
end;

procedure TformFileDlg.ShellListEdited(Sender: TObject; Item: TListItem;
  var S: string);
var
  OldIni, NewIni, OldFileName, NewFileName: string;
begin
  // S - отредактированное имя в дереве.
 NewFileName := ruIncludeTrailingBackslash(ShellTree.SelectedFolder.Path)
    + S + DefaultExt;
  OldFileName :=ruIncludeTrailingBackslash(ShellTree.SelectedFolder.Path)
    + OldName + DefaultExt;
  // С двойным расширением
  NewIni := NewFileName + '.ini';
  OldIni := OldFileName + '.ini';
  if FileExists(OldIni) then RenameFile(OldIni, NewIni);
  // На всякий случай для файлов с одинарным расширением
  OldIni :=
ChangeFileExt(ruIncludeTrailingBackslash(ShellTree.SelectedFolder.Path) +
    OldName, '.ini');
  NewIni := ChangeFileExt(NewFileName, '.ini');

  if FileExists(OldIni) then RenameFile(OldIni, NewIni);
  ruWriteDirinfoFileDescription(NewFileName, OldComment);
  ruWriteDirinfoFileDescription(OldFileName, 'Переименован в ' + S +
    DefaultExt);

end;

function TformFileDlg.NewFileNameValid(FileName: string): boolean;
var
  FileNameAndExt: string;
begin
  FileNameAndExt := ChangeFileExt(FileName, DefaultExt);
  Result := false;
  if not (ValidFileName(FileName)) then Exit;
  if MatchesMask(FileNameAndExt, FilterMask) then Result := true;
end;

procedure TformFileDlg.EditFileNameChange(Sender: TObject);
begin
```

```
 if CanCreateNewFile
  then btnOk.Enabled := NewFileNameValid(EditFileName.Text);
end;

procedure TformFileDlg.EditDescriptionChange(Sender: TObject);
begin
    EditDescription.Button.Enabled := (EditDescription.Text <> '');
end;

procedure TformFileDlg.EditDescriptionButtonClick(Sender: TObject);
{
Сохранение комментариев к папке или файлу
}
begin
if ActiveFolder then
  ruWriteDirinfoFolderDescription(ShellTree.SelectedFolder.Path,
    EditDescription.Text)
else
  ruWriteDirinfoFileDescription(ShellList.SelectedItem.Path,
     EditDescription.Text);
  EditDescription.Button.Enabled := False;
end;

procedure TformFileDlg.btnPropClick(Sender: TObject);
begin
{
Вывод диалога свойств файла или папки. Во многом ради того, чтобы так просто
вывести этот диалог, мы и применили нестандартные компоненты
}
  if ActiveControl = ShellList then ShellList.ShowPropertySheet
  else if ActiveControl = ShellTree then ShellTree.ShowPropertySheet;
end;

procedure TformFileDlg.btnCreateNewFolderClick(Sender: TObject);
// Создание новой папки
begin
  if not ShellTree.AddFolder('') then
    Application.MessageBox(PChar('Не могу создать папку '
     ), 'Ошибка!', MB_OK + MB_ICONHAND + MB_DEFBUTTON1
      + MB_APPLMODAL)
  else
    ShellTree.Refresh(ShellTree.Selected);
end;
procedure TformFileDlg.btnListClick(Sender: TObject);
// Изменение стиля отображения списка файлов
begin
  ShellList.ViewStyle := vsList;
end;

procedure TformFileDlg.btnDetailsClick(Sender: TObject);
// Изменение стиля отображения списка файлов
begin
```

```
  ShellList.ViewStyle := vsReport;
end;

procedure TformFileDlg.btnMoveUpClick(Sender: TObject);
// Переход в родительский каталог
begin
  ShellList.MoveUpOneLevel;
end;

procedure TformFileDlg.btnPreviewClick(Sender: TObject);
// Изменение режима отображения миниатюры
begin
  DWGpreview.Visible := btnPreview.Down;
end;

procedure TformFileDlg.btnSearchClick(Sender: TObject);
{
Поиск файлов по условию. Обратите внимание, что к диалогу поиска мы обращаемся
через COM-интерфейс, не включая модуль в состав приложения
}
var
  FileSearchDlg: IFileSearchDlg;
  i: integer;
  FileName: OleVariant;
  ResDial: OleVariant;
begin
  FileSearchDlg := CreateComObject(CLASS_FileSearchDlg) as
    IFileSearchDlg;
  FileSearchDlg.Create;
  FileSearchDlg.DlgCaption:=DlgCaption+ ' (поиск)';
  FileSearchDlg.RootDir:= ShellTree.SelectedFolder.Path;
  FileSearchDlg.FilterMask:=FilterMask;
  FileSearchDlg.MultiSelect:=MultiSelect;
  ResDial := FileSearchDlg.Execute;
  if ResDial then begin
{
Формирование списка найденных файлов
}
    StringListNames := TStringList.Create;
    for i := 0 to FileSearchDlg.FilesCount - 1 do
    begin
      FileName := FileSearchDlg.GetFile(i);
      StringListNames.Add(FileName);
    end;
{
Получив результаты, сразу выходим из приложения
}
    Close;
    ModalResult := mrok;
  end;
  FileSearchDlg.Free;
end;
```

```
procedure TformFileDlg.btnCancelClick(Sender: TObject);
// Выход по кнопке Отмена
begin
  Close;
  ModalResult := mrCancel;
end;

procedure TformFileDlg.btnOkClick(Sender: TObject);
// Выход по кнопке OK
var
  i: integer;
  FileName: string;
Begin
// Создаем список строк для результата выбора
  StringListNames := TStringList.Create;
  if CanCreateNewFile then
  begin
// В этом режиме всегда одиночный выбор
    FileName := EditFileName.Text;
    if ExtractFilePath(FileName) = '' then
      FileName :=
      ruIncludeTrailingBackslash(ShellTree.SelectedFolder.Path) +
        FileName;
// Устанавливаем расширение по умолчанию
    FileName := ChangeFileExt(FileName, DefaultExt);
// Добавляем имя файла в список результатов
    StringListNames.Add(FileName);
  end else if MultiSelect then
  begin
    for i := 0 to ShellList.SelectedItems.Count - 1 do
    begin
      if (not ShellList.SelectedItems[i].IsFolder) then
        StringListNames.Add(ShellList.SelectedItems[i].Path);
    end;
  end
  else
   StringListNames.Add(ShellList.SelectedItem.Path);
  Close;
  ModalResult := mrOk;
end;
end.
```

Функции для чтения и записи комментариев вынесены в отдельный модуль ruUtils (листинг 17.2). Полностью этот файл можно просмотреть на прилагаемом компакт-диске.

Листинг 17.2. Фрагмент модуля ruUtils.pas

```
function ruReadDirinfoFileDescription(FileName: string): string;
{
Чтение комментария к файлу FileName из файла dirinfo.ini. Если комментария нет,
возвращается имя файла без расширения
}
```

```
var
  IniFile: string;
  Ini: TIniFile;
begin
  Result := ChangeFileExt(ExtractFileName(FileName), '');
  IniFile := ExtractFilePath(FileName) + DirIniName;
  if FileExists(IniFile) then
  begin
    Ini := TIniFile.Create(IniFile);
    Result := Ini.ReadString(FileSecName, ExtractFileName(FileName),
      ChangeFileExt(ExtractFileName(FileName), ''));
    Ini.Free;
  end;
end;

function ruReadDirinfoFolderDescription(FolderName: string): string;
{
Чтение комментария к папке FolderName из файла dirinfo.ini.
}
var
  IniFile: string;
  Ini: TIniFile;
begin
  Result := '';
// Комментарий к папке записан в Dirinfo.ini родительской папки
  IniFile := ruIncludeTrailingBackslash(ruUpDir(FolderName)) +
   DirIniName;
  if FileExists(IniFile) then begin
    Ini := TIniFile.Create(IniFile);
    Result := Ini.ReadString(DirSecName, ExtractFileName(FolderName),'');
    Ini.Free;
  end;
end;

procedure ruWriteDirinfoFolderDescription(FolderName, Description: string);
// Запись комментария к папке
var
  IniFile: string;
  Ini: TIniFile;
begin
    IniFile := ruIncludeTrailingBackslash(ruUpDir(FolderName)) +
     DirIniName;
    Ini := TIniFile.Create(IniFile);
    Ini.WriteString(DirSecName, ExtractFileName(FolderName),
     Description);
    Ini.Free;
end;

procedure ruWriteDirinfoFileDescription(FileName, Description: string);
// Запись комментария к файлу
```

```
var
  IniFile: string;
  Ini: TIniFile;
begin
  IniFile := ExtractFilePath(FileName) + DirIniName;
  Ini := TIniFile.Create(IniFile);
  Ini.WriteString(FileSecName, ExtractFileName(FileName), Description);
  Ini.Free;
end;
```

Просмотр и редактирование расширенного набора свойств файлов не только в файловой системе NTFS

Иногда требуется не только просматривать, но и редактировать свойства DWG-файлов на компьютере с любой операционной системой, даже если система AutoCAD там "никогда не стояла". В разработанном файловом диалоге мы такую возможность не предусмотрели, т. к. наша система работает только совместно с AutoCAD. Но сделать такой вариант несложно. Закачиваем с сайта Autodesk файл

http://adeskftp.autodesk.com/prodsupp/autocad2000/dwgpro15.zip

В этом архиве находятся файлы pscan.cpp, DwgProps.h и DwgProps.obj. В первом файле имеется пример программы, извлекающей свойства DWG, а остальные два предназначены для использования при разработке программ. К сожалению, такие OBJ-файлы могут применяться только при использовании компиляторов Microsoft, напрямую прикомпоновать DwgProps.obj к Delphi-программе нельзя, поэтому придется написать обычную или ActiveX динамическую библиотеку на VC++ и использовать функции из этой библиотеки где угодно.

Создание COM-сервера

После отработки диалогового окна (сделать это можно, создав временный проект, в котором диалоговое окно будет использоваться в качестве главной формы) мы создадим COM-сервер для использования в любых приложениях. Делаем это в уже многократно отработанном порядке:

1. Создаем проект ActiveX DLL ruShellFileDlgSvr.dpr.
2. Создаем объект автоматизации `FileDialog`.
3. Редактируем библиотеку типов, описывая в ней свойства и методы объекта автоматизации.
4. В модуле реализации разрабатываем реализацию объявленных свойств и методов.

Фрагмент файла ruShellFileDlgSvr_TLB.pas приведен в листинге 17.3, модуль реализации — в листинге 17.4.

Листинг 17.3. Файл ruShellFileDlgSvr_TLB.pas

```
unit ruShellFileDlgSvr_TLB;

interface
```

```
uses Windows, ActiveX, Classes, Graphics, OleServer, OleCtrls, StdVCL;

const
  ruShellFileDlgSvrMajorVersion = 1;
  ruShellFileDlgSvrMinorVersion = 0;

  LIBID_ruShellFileDlgSvr: TGUID = '{995E0360-F4CC-11D7-8CF1-9E6E300F9D0D}';

  IID_IFileDialog: TGUID = '{995E0361-F4CC-11D7-8CF1-9E6E300F9D0D}';
  CLASS_FileDialog: TGUID = '{995E0363-F4CC-11D7-8CF1-9E6E300F9D0D}';
type

  IFileDialog = interface;
  IFileDialogDisp = dispinterface;

  FileDialog = IFileDialog;
  IFileDialog = interface(IDispatch)
    ['{995E0361-F4CC-11D7-8CF1-9E6E300F9D0D}']
    procedure Set_Caption(const Param1: WideString); safecall;
    procedure Set_RootDir(const Param1: WideString); safecall;
    procedure Set_StartDir(const Param1: WideString); safecall;
    procedure Set_FilterMask(const Param1: WideString); safecall;
    procedure Set_DefaultFileName(const Param1: WideString); safecall;
    procedure Set_MultiSelect(Param1: WordBool); safecall;
    procedure Set_CanCreateNewFile(Param1: WordBool); safecall;
    procedure Set_CanEdit(Param1: WordBool); safecall;
    function  Get_FilesCount: Integer; safecall;
    procedure Create; safecall;
    function  Execute: OleVariant; safecall;
    function  GetFile(Index: Integer): OleVariant; safecall;
    procedure Free; safecall;
    property Caption: WideString write Set_Caption;
    property RootDir: WideString write Set_RootDir;
    property StartDir: WideString write Set_StartDir;
    property FilterMask: WideString write Set_FilterMask;
    property DefaultFileName: WideString write Set_DefaultFileName;
    property MultiSelect: WordBool write Set_MultiSelect;
    property CanCreateNewFile: WordBool write Set_CanCreateNewFile;
    property CanEdit: WordBool write Set_CanEdit;
    property FilesCount: Integer read Get_FilesCount;
  end;

  IFileDialogDisp = dispinterface
    ['{995E0361-F4CC-11D7-8CF1-9E6E300F9D0D}']
    property Caption: WideString writeonly dispid 2;
    property RootDir: WideString writeonly dispid 3;
    property StartDir: WideString writeonly dispid 4;
    property FilterMask: WideString writeonly dispid 5;
    property DefaultFileName: WideString writeonly dispid 6;
    property MultiSelect: WordBool writeonly dispid 7;
    property CanCreateNewFile: WordBool writeonly dispid 8;
    property CanEdit: WordBool writeonly dispid 9;
```

```
    property FilesCount: Integer readonly dispid 11;
    procedure Create; dispid 12;
    function  Execute: OleVariant; dispid 13;
    function  GetFile(Index: Integer): OleVariant; dispid 14;
    procedure Free; dispid 15;
  end;

  CoFileDialog = class
    class function Create: IFileDialog;
    class function CreateRemote(const MachineName: string): IFileDialog;
  end;

implementation

uses ComObj;

class function CoFileDialog.Create: IFileDialog;
begin
  Result := CreateComObject(CLASS_FileDialog) as IFileDialog;
end;

class function CoFileDialog.CreateRemote(const MachineName: string): IFileDialog;
begin
  Result := CreateRemoteComObject(MachineName, CLASS_FileDialog)
   as IFileDialog;
end;
end.
```

Листинг 17.4. Файл u_ruShellFileDlgSvr.pas

```
unit u_ruShellFileDlgSvr;

interface

uses
  ComObj, ActiveX, ruShellFileDlgSvr_TLB, StdVcl;

type
  TFileDialog = class(TAutoObject, IFileDialog)
  protected
    function Get_FilesCount: Integer; safecall;
    procedure Create; safecall;
    function Execute: OleVariant; safecall;
    procedure Free; safecall;
    function GetFile(Index: Integer): OleVariant; safecall;
    procedure Set_CanCreateNewFile(Value: WordBool); safecall;
    procedure Set_CanEdit(Value: WordBool); safecall;
    procedure Set_Caption(const Value: WideString); safecall;
    procedure Set_DefaultFileName(const Value: WideString); safecall;
    procedure Set_FilterMask(const Value: WideString); safecall;
    procedure Set_MultiSelect(Value: WordBool); safecall;
    procedure Set_RootDir(const Value: WideString); safecall;
    procedure Set_StartDir(const Value: WideString); safecall;
```

```
    { Protected declarations }
  end;

implementation

uses ComServ, Controls, Forms, frmRuShellFileDlg;

function TFileDialog.Get_FilesCount: Integer;
begin
   Result := formFileDlg.StringListNames.Count;
end;

procedure TFileDialog.Create;
begin
   formFileDlg := TformFileDlg.Create(Application);
end;

function TFileDialog.Execute: OleVariant;
begin
   Result := (formFileDlg.ShowModal = mrOk);
end;

procedure TFileDialog.Free;
begin
   formFileDlg.Free
end;

function TFileDialog.GetFile(Index: Integer): OleVariant;
begin
  Result := formFileDlg.StringListNames.Strings[Index];
end;

procedure TFileDialog.Set_CanCreateNewFile(Value: WordBool);
begin
   formFileDlg.CanCreateNewFile := Value;
end;

procedure TFileDialog.Set_CanEdit(Value: WordBool);
begin
   formFileDlg.CanEdit := Value;
end;

procedure TFileDialog.Set_Caption(const Value: WideString);
begin
   formFileDlg.DlgCaption := Value;
end;

procedure TFileDialog.Set_DefaultFileName(const Value: WideString);
begin
   formFileDlg.DefaultFileName := Value
end;

procedure TFileDialog.Set_FilterMask(const Value: WideString);
begin
   formFileDlg.FilterMask := Value;
end;
```

```
procedure TFileDialog.Set_MultiSelect(Value: WordBool);
begin
    formFileDlg.MultiSelect := Value;
end;

procedure TFileDialog.Set_RootDir(const Value: WideString);
begin
   formFileDlg.RootDir := Value;
end;

procedure TFileDialog.Set_StartDir(const Value: WideString);
begin
  formFileDlg.StartDir := Value;
end;

initialization
  TAutoObjectFactory.Create(ComServer, TFileDialog, Class_FileDialog,
    ciMultiInstance, tmApartment);
end.
```

Пример использования разработанного диалогового окна выбора файлов в другой Delphi-программе можно найти в *главе 20*.

Формирование библиотеки функций для работы с файлами

Для использования диалогового окна выбора файлов в LISP-программах разработаем несколько функций. Сначала напишем низкоуровневую функцию, взаимодействующую с COM-сервером (листинг 17.5).

Листинг 17.5. Функция _ru-dlg-file

```
(defun _ru-dlg-file (dlg_caption root_dir start_dir filter_mask
 default_file_name is_multiselect is_can_edit is_can_create_new_file /
    srv result i)
;;;Диалог выбора файлов
 ;|
Аргументы:
dlg_caption - заголовок диалогового окна
root_dir - ограничивающий корневой каталог или ""
start_dir - начальный каталог
filter_mask - маска отображаемых файлов
default_file_name - имя нового файла по умолчанию
is_multiselect - возможность выбора группы файлов
is_can_edit - возможность редактирования комментариев
is_can_create_new_file - возможность возврата имени несуществующего файла
Пример 1
Выбор одного файла
(_ru-dlg-file "Выбор одного файла" "C:\\" "c:\\Data" "*.dwg"
  (vl-filename-base (getvar "DWGNAME")) nil T T)
("C:\\.ru\\cad\\samples\\dwg\\Слои\\Классификатор слоев.dwg")
```

```
Пример 2
Выбор нескольких файлов
(_ru-dlg-file "Выбор нескольких файлов" (getvar "DWGPREFIX")
  (getvar "DWGPREFIX") "*.dwg" "" T T T)

("C:\\.ru\\cad\\samples\\dwg\\Сантехника\\Водомеры.DWG"
  "C:\\.ru\\cad\\samples\\dwg\\Сантехника\\Sx_mag.dwg"
  "C:\\.ru\\cad\\samples\\dwg\\Сантехника\\demo-Водомер.dwg"
)
|;
 (if (setq srv (vlax-get-or-create-object
  "ruShellFileDlgSvr.FileDialog"))
  (progn
    (vlax-invoke-method srv "Create")
    (vlax-put-property srv 'Caption dlg_caption)
    (vlax-put-property srv 'RootDir root_dir)
    (vlax-put-property srv 'StartDir start_dir)
    (vlax-put-property srv 'FilterMask filter_mask)
    (vlax-put-property srv 'DefaultFileName default_file_name)
    (vlax-put-property srv'MultiSelect (ru-conv-value-to-wordbool
     is_multiselect))
    (vlax-put-property srv 'CanEdit (ru-conv-value-to-wordbool
     is_can_edit))
    (vlax-put-property srv 'CanCreateNewFile (ru-conv-value-to-wordbool
     is_can_create_new_file))
    (setq result (vlax-invoke-method srv "Execute"))
    (if (= (vlax-variant-value result) :vlax-true)
        (progn
          (setq result '() i 0)
          (repeat (vlax-get-property srv 'FilesCount)
            (setq result (cons (vlax-variant-value
            (vlax-invoke-method srv "GetFile" i)) result) i (1+ i))
         )
          (vlax-invoke-method srv "Free")
       );_ end of progn
        (setq result nil)
     );_ end of if
   );_ end of progn
    (princ "\nНе удалось запустить ruShellFileDlgSvr.FileDialog")
 );_ end of if
  result
)
```

Чаще всего нам придется выбирать DWG-файлы, поэтому напишем пару функций-оболочек (листинги 17.6 и 17.7), облегчающих разработку прикладных программ.

Листинг 17.6. Функция `ru-dlg-file-select-dwg`

```
(defun ru-dlg-file-select-dwg (dlg_caption start_dir root_dir is_edit
  is_multiselect)
```

```
;;; Выбор одного или нескольких существующих DWG-файлов
(_ru-dlg-file dlg_caption root_dir start_dir "*.dwg" "" is_multiselect
    is_edit nil)
)
```

Листинг 17.7. Функция ru-dlg-file-select-or-new-dwg

```
(defun ru-dlg-file-select-or-new-dwg (dlg_caption start_dir root_dir
  default_file_name is_edit)
;;; Выбор или создание одного DWG-файла
  (_ru-dlg-file dlg_caption root_dir start_dir "*.dwg" default_file_name
    nil is_edit t)
)
```

При разработке основного диалогового окна мы использовали диалоговый COM-сервер для поиска файлов по заданному условию и выбора одного или нескольких из списка найденных. Естественно, что такое диалоговое окно (см. рис. 17.3) мы можем оформить и в виде LISP-функции (листинг 17.8).

Листинг 17.8. Функция ru-dlg-file-search

```
(defun ru-dlg-file-search (dlg_caption root_dir filter_mask
 is_multiselect / srv result i)
;;;Диалог Поиска файлов
(if (setq srv (vlax-get-or-create-object
          "ruFileSearchSvr.FileSearchDlg"))
    (progn
      (vlax-invoke-method srv "Create")
      (vlax-put-property srv 'DlgCaption dlg_caption)
      (vlax-put-property srv 'RootDir root_dir)
      (vlax-put-property srv 'FilterMask filter_mask)
      (vlax-put-property srv 'MultiSelect
        (ru-conv-value-to-wordbool is_multiselect)
     );_ end of vlax-put-property
      (setq result (vlax-invoke-method srv "Execute"))
      (if (= (vlax-variant-value result) :vlax-true)
        (progn
          (setq result '() i 0)
          (repeat (vlax-get-property srv 'FilesCount)
            (setq result (cons (vlax-variant-value (vlax-invoke-method
              srv "GetFile" i)) result) i (1+ i))
         );_ end of repeat
          (vlax-invoke-method srv "Free")
       );_ end of progn
        (setq result nil)
     );_ end of if
   );_ end of progn
    (princ "\nНе удалось запустить ruFileSearchSvr.FileSearchDlg")
 );_ end of if
  result
)
```

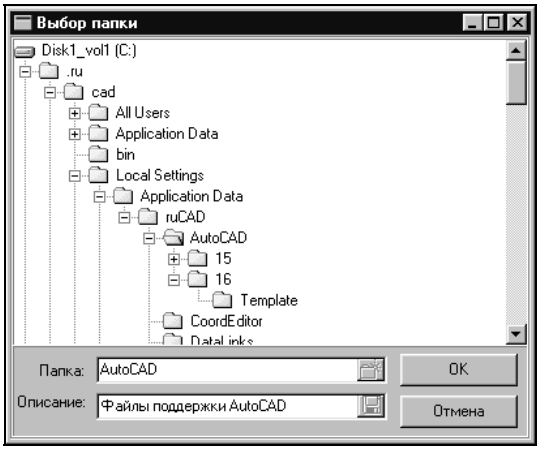


Рис. 17.4. Диалоговое окно выбора папки

Довольно часто требуется выбрать папку. При использовании стандартной функции `getfiled` приходится просить пользователя выбрать какой-нибудь файл в требуемой папке (очень "удобно", если папка пустая). Между делом пришлось разработать и диалоговое окно выбора папки (рис. 17.4). От стандартных оно отличается возможностью введения комментариев к папкам. Реализация в Delphi (разумеется в виде COM-сервера) настолько проста, что мы приведем только текст LISP-функции выбора папки (листинг 17.9).

Листинг 17.9. Функция _ru-dlg-folder

```
(defun _ru-dlg-folder (dlg_caption root_dir start_dir is_can_edit
                       / srv result)
;;;Диалог выбора папки
(if (setq srv (vlax-get-or-create-object "ruFolderDlgSvr.FolderDialog"))
    (progn
      (setq result (vlax-variant-value (vlax-invoke-method
           srv "SelectFolder" dlg_caption root_dir start_dir
                       (ru-conv-value-to-wordbool is_can_edit))))
      (if (= result "")(setq result nil))
   );_ end of progn
    (princ "\nНе удалось запустить ruFolderDlgSvr.FolderDialog ")
 );_ end of if
  result
)
```

В наших прежних разработках аналогичные функции возвращали списки выбранных файлов и комментариев к ним. Сначала казалось, что это хорошо, но практика применения показала, что комментарии чаще никак не используются, поэтому на уровне COM-серверов мы сделали возврат только имен файлов. Для того чтобы при необходимости получить или записать комментарий к файлу, разработаны две простые функции (листинги 17.10 и 17.11).

Листинг 17.10. Функция ru-file-read-dirinfo-file-comment

```
(defun ru-file-read-dirinfo-file-comment (file_name / srv result)
;;; Чтение комментария к файлу из dirinfo.ini
 (if (setq srv (vlax-get-or-create-object "ruFileInfoSvr.Dirinfo"))
    (vlax-invoke-method srv "ReadFileComment" file_name 'result)
    (princ "\nНе удалось запустить ruFileInfoSvr.Dirinfo")
 );_ end of if
  result
)
```

Листинг 17.11. Функция ru-file-write-dirinfo-file-comment

```
(defun ru-file-write-dirinfo-file-comment (file comment / srv)
;;; Запись комментария к файлу в dirinfo.ini
  (if (setq srv (vlax-get-or-create-object "ruFileInfoSvr.Dirinfo"))
    (vlax-invoke-method srv "WriteFileComment" file comment)
    (princ "\nНе удалось запустить ruFileInfoSvr.Dirinfo")
 );_ end of if
)
```

Как видите, даже в этих простых функциях использованы COM-технологии. И, честно говоря, мы даже забыли, что создали еще и такой COM-сервер. В нем содержится также функция получения свойств любого (не открытого в системе AutoCAD) DWG-файла (листинг 17.12). Останавливаться на реализации в Delphi у нас уже просто нет места. Намекнем только, что использованы полученные у фирмы Autodesk файлы DwgProps.h и DwgProps.obj.

Листинг 17.12. Функция _ru-dwgprops-read-dwg-file-info

```
(defun _ru-dwgprops-read-dwg-file-info (file_name / srv result i
                                        name value)
;;; Получение свойств постороннего DWG-файла
 ;|
(_ru-dwgprops-read-dwg-file-info "c:\\.ru\\cad\\samples\\dwg\\Сантехника
\\demo-Водомер.dwg")
(
  ("DirinfoComment" . "Водомерный узел № 3")
  ("Title" . "Водомерный узел № 2")
  ("Author" . "Гипроавтоагрегат. ОТЭ. Исп Зуев С.А. ГИП Корляков Н.Л.")
  ("Comments" ."Дополнительный водомерный узел №2 корпуса 15 КЗКТ. Вариант
с обводной линией.")
  ("Keywords" . "КЗКТ водоснабжение корпус 15")
  ("Subject" . "КЗКТ. Корпус 15. 3 очередь")
  ("HyperlinkBase" . "")
  ("RevisionNo" . "238.00.01")
  ("LastSavedBy" . "Зуев С.А. ©, DataCAD Group")
  ("Indwg" . "21h 51m")
  ("Created" . "18 марта 2001 г.  20:18:10")
  ("Updated" . "7 июня 2003 г.  15:27:33")
```

```
  ("AcadDWGCustom1" . "М 1:=10")
  ("AcadDWGCustom2" . "Ед. изм=мм")
  ("AcadDWGCustom3" . "Тчк. вст=0.5 мм на бумаге")
  ("AcadDWGCustom4" . "Программист=2.5 мм на бумаге")
  ("AcadDWGCustom5" . "Админ=qqqqqqqqqqqqqqqq")
  ("AcadDWGCustom6" . "Помощник=Нет")
  ("AcadDWGCustom7" . "Чайник=Нет")
  ("AcadDWGCustom8" . "Админ=Да")
  ("AcadDWGCustom9" . "Помощник=Нет")
  ("AcadDWGCustom10" . "Чайник=Нет")
)
|;
  (if (setq srv (vlax-get-or-create-object "ruFileInfoSvr.DwgFileInfo"))
    (progn
      (vlax-invoke-method srv "Create")
      (vlax-invoke-method srv "GetFileInfo" file_name)
      (setq i 0)
      (repeat (vlax-get-property srv 'PropsCount)
        (vlax-invoke-method srv "GetProperty" i 'value 'name)
        (setq result (cons (cons  name  value) result) i (1+ i))
     );_ end of repeat
      (vlax-invoke-method srv "Free")
   );_ end of progn
    (princ "\nНе удалось запустить ruFileInfoSvr.DwgFileInfo")
 );_ end of if
  (reverse result)
);_ end of defun
```

Итак, теперь мы можем использовать в своих программах достаточно удобные диалоговые окна выбора папок и файлов и получать аннотирующую информацию. А это уже может служить начальной базой для системы документооборота.

ГЛАВА 18

Разработка классификатора слоев

О необходимости создания классификатора слоев мы писали в *главе 4*. Там же были изложены основные правила именования слоев и способы хранения классификатора (база данных на удаленном SQL-сервере, использование клиентских наборов данных на локальных компьютерах, периодически синхронизируемых с базой данных на удаленном SQL-сервере, базы данных на локальных компьютерах в формате Microsoft Access или XML).

За десять лет работы с классификаторами слоев мы испробовали множество вариантов. Общее у них то, что в любом варианте желательно визуальное отображение иерархии слоев в виде разворачивающегося дерева.

Как реализовать классификатор слоев с использованием XML

Очень легко реализовать классификатор слоев в формате XML — точно так же, как мы реализовали работу с деревом меню. Не надо даже писать дополнительных программ для обработки дерева классификатора, достаточно предусмотреть соответствующие атрибуты и обработку на уровне LISP. Например, один из вложенных XML-файлов общего классификатора может выглядеть так (листинг 18.1).

Листинг 18.1. Фрагмент классификатора слоев в виде XML-документа

```
<?xml version='1.0' encoding='windows-1251'?>
<Layers name='Слои'>
   <РП_Общие_ru name='Общие'>
      <РП_Общие_Заглавные_ru name='Заглавные'>
         <РП_Общие_Заглавные_Формат_ru name='Формат'
             LayerAltName='03091821480995'
             LayerComment='' LayerColor='7'
             LayerIni='\РП\Общие\Заглавные\Формат.ini'
             LayerMethod='(ru-app-load "ru_draw_format")'
             LayerMethodComment='Рисование формата с помощью мастера'
             LayerUseAltName='0'
             LayerIsSpecial='0'/>
```

```
      </РП_Общие_Заглавные_ru>
     </РП_Общие_ru>
</Layers>
```

Выбор из такого классификатора осуществляется точно так же, как и из XML-меню. Дополнительные данные по слою хранятся в INI-файле.

Хранение классификатора в базе данных

При хранении классификатора в базе данных любого формата для отображения таблицы слоев в виде дерева применяется стандартный прием для отображения иерархии в различных компонентах, обычно именуемых `xxDbTreeView`:

- создается поле с уникальным номером записи, например с именем `ID`;
- создается поле с номером записи "родителя", например с именем `ID_PARENT`.

При использовании тех или иных `xxDbTreeView` настраиваются свойства компонента (источник данных, имена полей и пр.) и иерархия базы данных отображается в виде дерева. В зависимости от "навороченности" `xxDbTreeView` происходит автоматическое изменение полей `ID` и `ID_PARENT` при добавлении или перетаскивании узлов дерева.

Опыт работы показывает, что хранение фиксированных имен в том или ином хранилище имеет (наряду с массой достоинств) существенные недостатки.

Редактирование базы данных нельзя доверять каждой "тетке", а слоев требуется очень много, причем очень много однотипных слоев для каждого раздела. Например, все разделы могут иметь слои со "своими" размерами и надписями, которые при компоновке сборного рисунка желательно не перепутать. Это значит, что не должно быть слоя РАЗМЕРЫ, а должны быть РАЗМЕРЫ_АР, РАЗМЕРЫ_ОВ, РАЗМЕРЫ_ЕО, да еще, возможно, для каждого этажа, для каждой стадии проектирования. Кто-то должен создать в классификаторе слои, допустим, для десятка разделов и шестнадцати этажей. Практика показывает, что "центральную" базу классификатора, да еще в современных экономических условиях редактировать просто некому, а имеющаяся не удовлетворяет пользователей. Классификатор живет сам по себе, а пользователи продолжают создавать слои как попало.

Классификатор слоев в файловой системе

Для решения этой проблемы мы решили вернуться к использовавшейся несколько лет назад практике хранения классификатора в системе каталогов.

Суть этой технологии:

- в некотором корневом каталоге, например, %RuCadRootDir%\All Users\Layers\ru\ (именуемом в дальнейшем %RuCADLayerRootDir%) создается специальная система подкаталогов;
- имя каждого подкаталога является частью имени слоя;
- в любом подкаталоге могут находиться файлы с расширением lay, в которых в INI-формате может храниться любая информация по слою;

- базовое имя LAY-файла является последним "подсловом" в имени слоя;
- имя каждого слоя формируется автоматически путем замены в полном имени LAY-файла символов-разделителей каталогов на заданный разделитель "подслов" с исключением имени корневого каталога классификатора.

Таким образом, файлу *%RuCADLayerRootDir%\РП\План\1эт\Размеры\Ac.lay* будет соответствовать слой *РП_План_1эт_Размеры_Ac*.

Рядом с файлом Ac.lay могут располагаться файлы OB.lay, BK.lay и др. В результате в классификаторе будут присутствовать слои *РП_План_1эт_Размеры_OB* и *РП_План_1эт_Размеры_BK*. Кроме того, может существовать и файл *%RuCADLayerRootDir%\РП\План\1эт\Размеры.lay* и соответствующий ему слой *РП_План_1эт_Размеры* для каких-то общих размеров этажа, а также файл *%RuCADLayerRootDir%\РП\План\Размеры.lay* и соответствующий ему слой *РП_План_Размеры* для общих размеров здания.

При необходимости можно скопировать наборы LAY-файлов, созданных в одном каталоге, в другие подкаталоги, например из подкаталога *1эт* в подкаталоги *2эт—16эт*. Система каталогов и их имен может быть любой, удобной для проектной организации. К именам слоев может автоматически добавляться фирменный префикс или суффикс.

В LAY-файлах в виде переменных описываются основные параметры слоя — альтернативное имя (на другом языке или в виде короткого кода), комментарий к слою, цвет слоя на экране, цвет слоя на бумаге, тип линии, вес линии и другие параметры по умолчанию. Для слоя может быть задан метод создания объектов. Обычно это загрузка и выполнение конкретной программы, но может быть и загрузка фрагмента XML-меню с соответствующими программами. Например, для слоя координационных осей здания это может быть программа рисования осей, а для слоя трубопроводов — загрузка XML-меню программ рисования трубопроводов. Пример LAY-файла показан в листинге 18.2.

Листинг 18.2. Пример файла описания слоя Формат.LAY

```
[LayerParams]
LayerIsSpecial=0
LayerColor=9
LayerComment=Слой для рисования форматов чертежей
LayerMethod=(ru-app-load "ru_draw_format")
LayerMethodComment=Рисование формата с помощью мастера
LayerAltName=TILE_BLOCK_RU
LayerLineweight=0
```

Какие дополнительные данные можно хранить в классификаторе

Хранение описания слоя в каталоге позволяет воспользоваться преимуществами файловой системы как хранилища информации. В каталоге можно хранить любые файлы, относящиеся к слою. Пользователям САПР это, возможно, и не потребуется, но для создания информационных систем требуется дополнительная информа-

ция, связанная со слоем. В нашей практике создания геоинформационных систем такими дополнительными данными являлись:

- иллюстрации к слоям (это пригодится и в САПР);
- структуры баз данных и SQL-скрипты для создания баз данных;
- справочники, облегчающие ввод данных;
- пиктограммы информационных объектов в виде блоков.

В каталоге слоя, кроме прочего, можно хранить и сами слои в виде DWG-файлов. В этом случае классификатор одновременно является и своеобразным архивом. Многие проектные организации занимаются так называемым *генпроектированием*, т. е. разрабатывают в течение ряда лет проекты какого-то предприятия. В этом случае часть классификатора можно построить и по "территориальному" принципу, т. е. создавать слои для конкретных объектов. Можно, например, создать ветвь каталога %RuCADLayerRootDir%\ЗИЛ\Корпуса\15\План\1эт и складывать в этот каталог, кроме описания слоев, и соответствующие им "кальки", которые впоследствии можно будет загружать в виде ссылок или вставлять в сборный рисунок.

Разумеется, все подобные действия легко запрограммировать, а как это сделать, мы рассмотрим далее.

Конечно, размещение классификатора в файловой системе имеет и недостатки. Здесь очень важную роль играет квалификация системного администратора. Внутри нашей системы также будут предусмотрены некоторые меры безопасности — не всякий пользователь сможет программно изменять данные классификатора, а о том, чтобы он физически не мог проникнуть в определенные каталоги, должен позаботиться "злобный сисадмин".

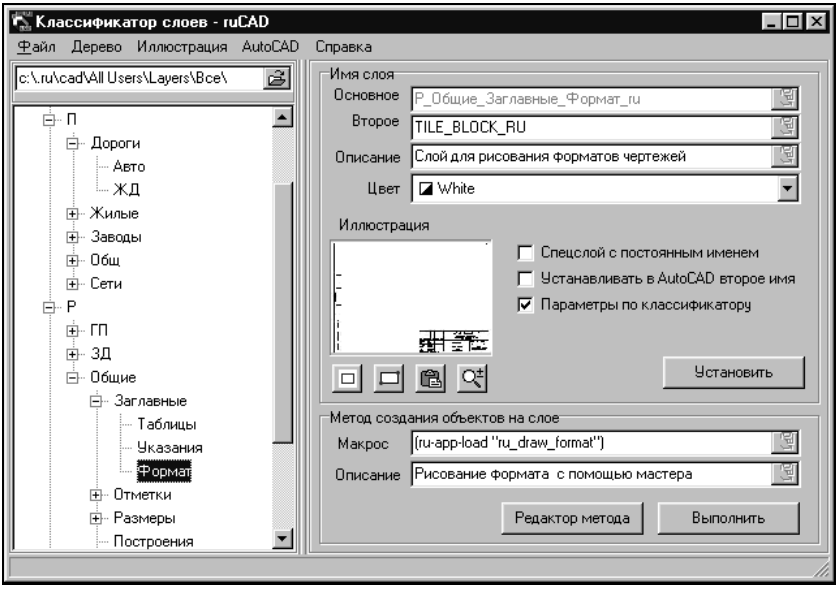


Рис. 18.1. Программа работы с классификатором слоев

Если проектная организация "доросла" до "вышей формы" хранения классификатора в базе данных на удаленном сервере, то будет очень просто преобразовать "устоявшийся" классификатор из файловой системы в любую базу данных.

Внешний вид программы для работы с классификатором представлен на рис. 18.1.

Работа с классификатором

Сформулируем основные требования к программе, которые в дальнейшем постараемся реализовать.

1. Классификатор слоев, в отличие от множества других разработанных нами приложений, удобнее использовать в виде самостоятельного приложения, а не DLL-сервера. Обращения к классификатору должны происходить очень часто, в идеале любой слой должен создаваться только с использованием классификатора. Запуск такого довольно "тяжелого" приложения требует некоторого времени (пусть 1—3 сек.), и постоянная его перегрузка раздражает пользователей. Если же приложение-классификатор работает параллельно с AutoCAD, переключение между ними не составляет труда, дерево слоев постоянно развернуто в нужном месте, и для установки классифицированного слоя достаточно щелкнуть по кнопке.
2. Классификатор должен уметь обнаруживать запущенную систему AutoCAD, присоединяться к AutoCAD любой из поддерживаемых версий. Классификатор должен уметь установить слой с заданным (основным или альтернативным) именем и заставить AutoCAD выполнить программу-метод для выбранного слоя.
3. Дерево слоев должно перечитываться достаточно быстро, в идеале не перечитывая весь классификатор, а только по мере раскрытия ветвей.
4. Некоторые слои могут быть объявлены *специальными*. К специальным относятся служебные слои AutoCAD ("0", "DEFPOINTS" и т. п.) и других программ, создающих слои с фиксированными именами.
5. Параметры слоев должны достаточно легко настраиваться и изменяться.
6. Классификатор должен уметь экспортировать дерево слоев в форматы общего применения.

Работа обычных пользователей с программой-классификатором заключается в следующем:

- производится запуск классификатора из меню системы AutoCAD;
- при необходимости установки классифицированного слоя его "человеческое" имя выбирается в дереве;
- щелчком по кнопке **Установить** в системе AutoCAD текущим устанавливается выбранный слой, при этом:
  - если выбран флажок **Параметры по классификатору**, то устанавливается цвет слоя по классификатору (при этом может произойти регенерация рисунка, если пользователь средствами AutoCAD установил другой цвет);
  - если выбран флажок **Устанавливать в AutoCAD второе имя**, то устанавливается альтернативное имя слоя.

- щелчком по кнопке **Выполнить** в AutoCAD посылается соответствующий макрос — обычно это LISP-выражение, хотя может быть и любое выражение, воспринимаемое командной строкой системы AutoCAD.

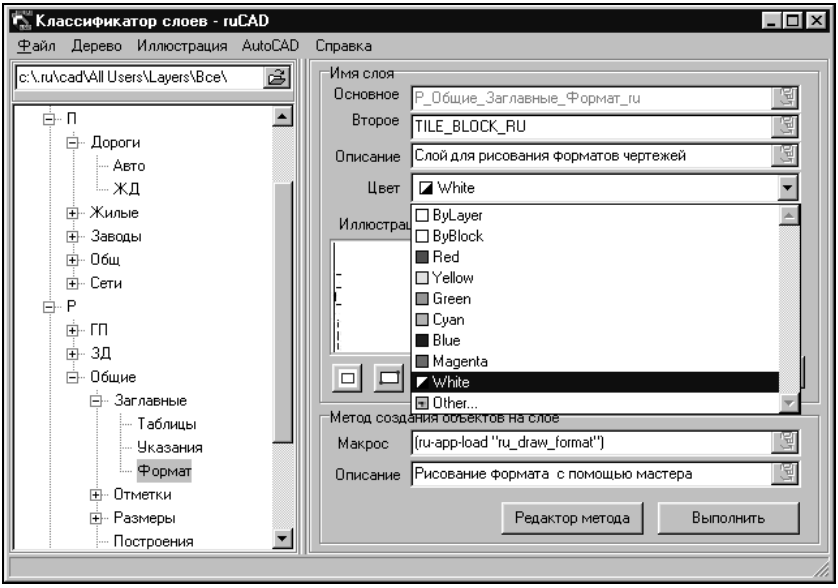


Рис. 18.2. Редактор классификатора при выборе цвета слоя

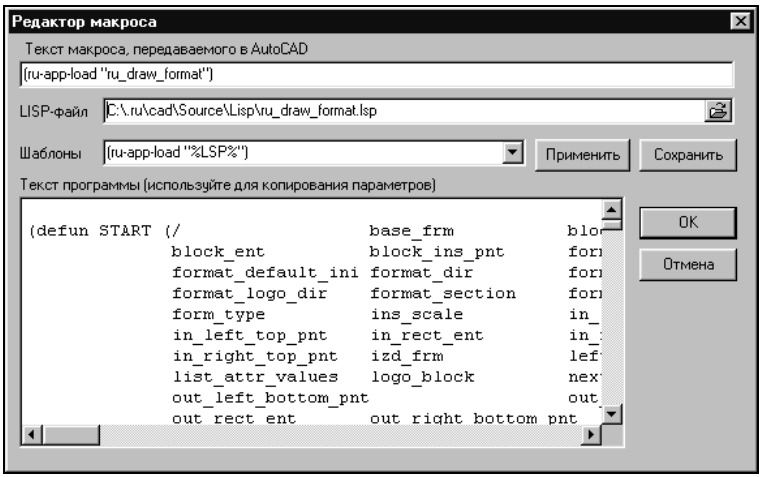


Рис. 18.3. Редактор метода рисования для слоя

Квалифицированный пользователь может выполнять и другие действия по настройке классификатора: добавлять ветви и слои, настраивать цвет (рис. 18.2), конструировать макрос слоя (рис. 18.3) или просто вписывать его в поле редактирования,

создавать иллюстрации к слою вставкой из буфера обмена или вырезкой с экрана, редактировать имена слоев и описания. Для сохранения изменений в каждом поле редактирования предусмотрена кнопка записи.

При редактировании макроса (см. рис. 18.3) в специальном диалоговом окне можно воспользоваться типовым шаблоном, выбираемым из списка, выбрать и просмотреть LISP-программу (если доступны исходные тексты), просмотреть или скопировать в текст макроса аргументы функции. В случае применения шаблона в заготовку макроса вставляется имя выбранного LISP-файла. Конечно, это работа не для "теток" и не для "чайников".

Разработка программы

Теперь, зная поставленную задачу, мы сможем разработать программу. С учетом изложенных требований в качестве основного визуального компонента классификатора — дерева слоев мы будем использовать замечательный компонент `VirtualTreeView`[1]. Это "навороченное до ужаса" дерево имеет множество свойств и методов, позволяющих очень быстро строить иерархические деревья любой информации, с любыми настройками отображения. Особенно привлекает то, что дерево может строиться не сразу, а частями, по мере обращения пользователей к узлам. Это именно то, что нам нужно, т. к. перечитывание большого количества элементов файловой системы обычными методами занимает много времени. Так как никакие "украшения" в виде фонов, пиктограммы узлов, нестандартных шрифтов мы использовать не намерены, воспользуемся самым простым и быстрым классом `TVirtualStringTree`.

Вторым интересным моментом является выбор цвета слоя. В Delphi используется множество компонентов для выбора цвета, но все они позволяют получить номер цвета, используемого в Windows, а не в системе AutoCAD, в которой применяется набор цветов *ACI* (AutoCAD Color Index)[2]. Аналитический перевод цветов Windows (OLE Color) в ACI мы пока не встречали — производится простое сопоставление номера цвета из OLE Color наиболее подходящему номеру цвета из ACI, что очень ненадежно, т. к. дает разные результаты для различных режимов монитора[3]. Конечно, немного потрудившись, можно сделать и соответствующий компонент для Delphi, но времени у нас нет, и мы, руководствуясь принципом максимального использования готовых решений, применим для выбора цвета AutoCAD ActiveX Control AcadColor.ocx от Cyrille Fauvel, Developer Consulting Group, Autodesk[4]. Этот ActiveX Control полностью имитирует раскрывающийся список и диалоговое окно выбора цвета в системе AutoCAD. Поскольку "The 'AcadColor.ocx' ActiveX control is also subject to the Shareware restriction use but is free of charge for ADN members", мы, как только выберется свободное время, все-таки сделаем свой бесплатный аналог.

---

[1] **www.delphi-gems.com**.

[2] В системе AutoCAD 2004 введен еще цвет `TrueColor`, использующий RGB-компоненты. Он значительно расширил палитру цветов.

[3] В AutoCAD 2004 появилась специальная библиотека для работы с цветом, но нам-то нужно обеспечить и работу с AutoCAD 2002.

[4] **www.microtec.net/~mrochon/files/acadcolor.zip**.

А пока мы импортируем библиотеку типов AcadColor.tlb, положим AcadColor.ocx в свой каталог Bin и зарегистрируем из командной строки:

```
Regsvr32 AcadColor.ocx
```

Теперь мы окончательно можем приступить к разработке приложения ruLayersExplorer. Ради традиционной экономии места текст формы приводить не будем. Немного сокращенный за счет малозначительных подробностей модуль главной формы приведен в листинге 18.3, а основной модуль — в листинге 18.4. По традиции важные участки кода выделяем полужирным шрифтом.

Листинг 18.3. Файл frmTreeLayer.pas

```
unit frmTreeLayer;
interface

uses
  SysUtils, Windows, Messages, Classes, Controls, Forms, Dialogs, Menus, ActnList,
ComCtrls, StdCtrls, ExtCtrls, Mask, JvToolEdit, JvCombobox,
  GR32_Image, Buttons, ruTxtViewSrv_TLB, JvComponent,
 ToolWin, JvToolBar, ImgList, ExtDlgs, OleCtrls, AutoCADColor_TLB,
 janXMLparser2, VirtualTrees, ruOnlyOpenOnce;
type

  TruTreeLayer = class(TForm)
    StatusBar: TStatusBar;
    MainMenu1: TMainMenu;
    miFile: TMenuItem;
    miFileExit: TMenuItem;
    mnuTree: TMenuItem;
    MenuItem2: TMenuItem;
    mnuImage: TMenuItem;
    miMakeImage: TMenuItem;
    miPasteFromClip1: TMenuItem;
    miPasteImageFree1: TMenuItem;
    miPasteStandardImage1: TMenuItem;
    miAutoCAD: TMenuItem;
    miSendAcad: TMenuItem;
    mnHelp: TMenuItem;
    miAbout: TMenuItem;
    ActionList1: TActionList;
    aPasteFromClip: TAction;
    aPasteStandardImage: TAction;
    aPasteImageFree: TAction;
    aShowFullImage: TAction;
    aExpandTree: TAction;
    aCollapse: TAction;
    aExit: TAction;
    aTreeAdd: TAction;
    aSendToAcad: TAction;
    aEditMacro: TAction;
    aAbout: TAction;
```

```
    aSetLayerInAcad: TAction;
    ImageList1: TImageList;
    GroupBoxName: TGroupBox;
    CheckBoxUseAltName: TCheckBox;
    CheckBoxSpecLayer: TCheckBox;
    GroupBoxMethod: TGroupBox;
    EditLayerMethod: TJvComboEdit;
    btnSetLayerInAcad: TButton;
    btnSendToAcad: TButton;
    AutoCAD1: TMenuItem;
    CheckBoxParamsByClass: TCheckBox;
    AcadColor: TAcadColor;
    miFileReload: TMenuItem;
    aTreeAddChild: TAction;
    aReload: TAction;
    aRenameLayer: TAction;
    PopupMenuTree: TPopupMenu;
    SpeedButtonPasteStandardImage: TSpeedButton;
    SpeedButtonPasteImageFree: TSpeedButton;
    SpeedButtonPasteFromClip: TSpeedButton;
    SpeedButtonShowFull: TSpeedButton;
    PanelImageBack: TPanel;
    Image: TImage32;
    Label5: TLabel;
    aExportToXml: TAction;
    miExportXML: TMenuItem;
    JvDirectoryEdit: TJvDirectoryEdit;
    TreeDir: TVirtualStringTree;
    EditLayerName: TJvComboEdit;
    EditLayerAltName: TJvComboEdit;
    EditLayerComment: TJvComboEdit;
    Button1: TButton;
    EditLayerMethodComment: TJvComboEdit;
// Компонент, разрешающий запуск только одного экземпляра приложения
    OnlyOneApp: TruOnlyOpenOnce;
    procedure FormCreate(Sender: TObject);
    procedure ReadLayerParams(CurrentFullPath: string);
    function FileNameToTreePath(FileName, Ext: string): string;
    function TreePathToFileName(TreePath: string): string;
    function LayerFileNameToLayerName(FileName: string): string;
    function RelativeLayerFileName(LayerFileName: string): string;
    procedure AddLayerToTree(ChildLayer: boolean);
    function InputNewLayerName: string;
    procedure ActionList1Update(Action: TBasicAction;
      var Handled: Boolean);
    procedure FormShow(Sender: TObject);
    procedure AcadColorSelChange(Sender: TObject; wParam,
        lParam: Integer);
    procedure CheckBoxSpecLayerClick(Sender: TObject);
    procedure CheckBoxUseAltNameClick(Sender: TObject);
    procedure aExpandExecute(Sender: TObject);
```

```
  procedure aCollapseExecute(Sender: TObject);
  procedure aPasteFromClipExecute(Sender: TObject);
  procedure aPasteStandardImageExecute(Sender: TObject);
  procedure aPasteImageFreeExecute(Sender: TObject);
  procedure aShowFullImageExecute(Sender: TObject);
  procedure aExitExecute(Sender: TObject);
  procedure aTreeAddExecute(Sender: TObject);
  procedure aSendToAcadExecute(Sender: TObject);
  procedure aEditMacroExecute(Sender: TObject);
  procedure aAboutExecute(Sender: TObject);
  procedure aSetLayerInAcadExecute(Sender: TObject);
  procedure aReloadExecute(Sender: TObject);
  procedure aTreeAddChildExecute(Sender: TObject);
  procedure aExportToXmlExecute(Sender: TObject);
  procedure SaveToXML;
  procedure GenerateXML(StartDomNode: TjanXMLNode2;
            StartTreeNode: PVirtualNode);
  procedure MakeDomNodeAttribs(CurrDomNode: TjanXMLNode2; CurrTreeNode:
    PVirtualNode);
  procedure MakeLayerVars(NodePathString: string; DeleteRoot: boolean);
  procedure JvDirectoryEditAfterDialog(Sender: TObject;
   var Name: string; var Action: Boolean);
  procedure LoadDir(DirName: string);
  procedure TreeDirChange(Sender: TBaseVirtualTree;
      Node: PVirtualNode);
  procedure TreeDirCompareNodes(Sender: TBaseVirtualTree; Node1,
    Node2: PVirtualNode; Column: TColumnIndex; var Result: Integer);
  procedure TreeDirFreeNode(Sender: TBaseVirtualTree;
    Node: PVirtualNode);
  procedure TreeDirGetText(Sender: TBaseVirtualTree;
    Node: PVirtualNode; Column: TColumnIndex; TextType: TVSTTextType;
    var CellText: WideString);
  procedure TreeDirInitChildren(Sender: TBaseVirtualTree;
    Node: PVirtualNode; var ChildCount: Cardinal);
  procedure TreeDirInitNode(Sender: TBaseVirtualTree; ParentNode,
    Node: PVirtualNode; var InitialStates: TVirtualNodeInitStates);
  procedure EditLayerNameButtonClick(Sender: TObject);
  procedure EditLayerAltNameButtonClick(Sender: TObject);
  procedure EditLayerCommentButtonClick(Sender: TObject);
  procedure EditLayerCommentChange(Sender: TObject);
  procedure EditLayerNameChange(Sender: TObject);
  procedure EditLayerAltNameChange(Sender: TObject);
  procedure EditLayerMethodChange(Sender: TObject);
  procedure EditLayerMethodButtonClick(Sender: TObject);
  procedure EditLayerMethodCommentButtonClick(Sender: TObject);
  procedure EditLayerMethodCommentChange(Sender: TObject);
  procedure OnlyOneAppNewInstance(Sender: TObject);

private
  AcadFound, SlowExpanded, LoadOnShow: boolean;
  LayerUseAltName, LayerIsSpecial, LayerParamsEdited: boolean;
  LayerImageFile, LayerAltName, LayerMethod, LayerMethodComment,
```

```
    LayerComment, LayerName, LayerFileName, LayerShortIni, LayerDir,
    TreeRootDir: string;
    LayerColor: integer;
  public
    StartIniFile: string;
    AcadAppString, AboutFileName,
    StorageIniFile, XMLImagesDir, XMLRootDir: string;
  end;
var
  ruTreeLayer: TruTreeLayer;

implementation

uses Graphics, ruUtils, ruAcad, ruSplash, JclStrings, JclFileUtils,
 IniFiles,Dim, janStrings, ruConst, ruTreeXMLUtils, ComObj, uCaptureForm,
  frmMacroEdit, ShlObj, ActiveX, ruFileUtils, QStrings, ruTreeDirUtils,
  ruTreeLayerUtils;

{$R *.DFM}
const
  NewNumber: integer = 0;

var
{
Мы объявляем объект AcadApplication как OleVariant, а не как IacadApplication и не
импортируем библиотеку типов. Подробно причины такого поступка описаны в главе 20.
}
  AcadApplication: OleVariant;
  About: ITxtFileViewer;
  sCmdLine: string;

procedure TruTreeLayer.OnlyOneAppNewInstance(Sender: TObject);
begin
{ Предотвращение запуска последующих экземпляров приложения. При попытке запуска
только восстанавливается окно уже работающей программы
}
  if WindowState = wsMinimized then WindowState := wsNormal;
end;

procedure TruTreeLayer.FormCreate(Sender: TObject);
begin
{ Предотвращение запуска последующих экземпляров приложения. При попытке запуска
только восстанавливается окно уже работающей программы
}
 if OnlyOneApp.ProcTerminatingApplication then Exit;
{
 Очень важно! Имя объекта автоматизации задается не "AutoCAD.Application",
"AutoCAD.Application.15" или "AutoCAD.Application.16", а извлекается из реестра,
куда было записано программой-стартером. Подробности в главе 20
}
// ====================================
    AcadAppString:= ruGet_AcadAppString;
// ====================================
  AcadFound := False;
```

```
// Попытка подключения к AutoCAD
 try
      AcadApplication := GetActiveOleObject(AcadAppString);
      AcadFound := True;
    except
     AcadFound := False;
     MessageDlg('Не найден AutoCAD ('+ AcadAppString + ')!', mtError,
     [mbOK], 0);
 end;
 // Факультативный параметр - имя корневого каталога
   sCmdLine := ruGetParameter;
  if sCmdLine <> '' then
    LayerDir := ruExcludeTrailingBackslash (sCmdLine)
 else LayerDir := ruGet_LayersClassDir;
  TreeRootDir := LayerDir;
  LoadOnShow := True;
  SlowExpanded := True;
  Constraints.MaxHeight := Screen.Height;
  Constraints.MaxWidth := Screen.Width;
  SpeedButtonShowFull.Caption := '';
  SpeedButtonPasteStandardImage.Caption := '';
  SpeedButtonPasteImageFree.Caption := '';
  SpeedButtonPasteFromClip.Caption := '';
  StatusBar.SimpleText := '';
  LayerName := '';
  LayerImageFile := '';
  LayerParamsEdited := False;
  JvDirectoryEdit.InitialDir := TreeRootDir;
  JvDirectoryEdit.Text := TreeRootDir;
  XMLImagesDir := ruGet_XmlImagesDir;
  StorageIniFile := IncludeTrailingBackslash(ruGet_LocalAppDataDir) +
    'LayerClassExplorer\FormStorage.ini';
  AboutFileName := IncludeTrailingBackslash(ruGet_LocalAppDataDir) +
    'LayerClassExplorer\About.txt';
  About := CreateComObject(CLASS_TxtFileViewer) as ITxtFileViewer;
{ Инициализация дерева слоев
Процедура вынесена в модуль ruTreeDirUtils (листинг 18.4). В ней нет ничего,
кроме выделения памяти для данных, привязываемых к узлам дерева, и установки одного
корневого узла. А где же построение дерева, наподобие того, которое мы делали
в главе 16 при построении дерева XML-документа?
Суть же в том, что VirtualTree не строится полностью сразу (это произойдет только
при выполнении метода ExpandAll), а строит отдельные ветви при раскрытии их
пользователем
}
  ruTreeDirInit(TreeDir);
end;
procedure TruTreeLayer.FormShow(Sender: TObject);
begin
  ActiveControl := TreeDir;
end;
```

```
procedure TruTreeLayer.LoadDir(DirName: string);
{
Перезагрузка дерева с заданным корневым каталогом
}
begin
  TreeRootDir := DirName;
  SlowExpanded := True;
  TreeDir.Clear;
  ruTreeDirInit(TreeDir);
  StatusBar.SimpleText := '';
end;

{Далее группа процедур и функций с именами TruTreeLayer.TreeDirXXXX,
в которых и скрыта работа с деревом
}

procedure TruTreeLayer.TreeDirCompareNodes(Sender: TBaseVirtualTree;
  Node1, Node2: PVirtualNode; Column: TColumnIndex; var Result: Integer);
{
Сравнение узлов для того, чтобы узлы с папками показывать выше узлов с файлами
}
begin
  Result := ruTreeDirCompareNodes(Sender, Node1, Node2);
end;

procedure TruTreeLayer.TreeDirFreeNode(Sender: TBaseVirtualTree;
  Node: PVirtualNode);
{
Освобождение памяти, зарезервированной для узла
}
begin
  ruTreeDirFreeNode(Sender, Node);
end;

procedure TruTreeLayer.TreeDirGetText(Sender: TBaseVirtualTree;
  Node: PVirtualNode; Column: TColumnIndex; TextType: TVSTTextType;
  var CellText: WideString);
{
Отображение текста узла.
}
begin
  CellText := ruTreeDirGetText(Sender, Node);
end;

procedure TruTreeLayer.TreeDirInitChildren(Sender: TBaseVirtualTree;
  Node: PVirtualNode; var ChildCount: Cardinal);
{
Инициализация узла-потомка
}
begin
  ruTreeDirInitChildren(Sender, Node, ChildCount, LayerExt);
end;
```

```
procedure TruTreeLayer.TreeDirInitNode(Sender: TBaseVirtualTree;
  ParentNode, Node: PVirtualNode;
  var InitialStates: TVirtualNodeInitStates);
{
Инициализация узла
}
begin
  ruTreeDirInitNode(Sender, ParentNode, Node, InitialStates, {LayerDir}
    TreeRootDir);
end;

procedure TruTreeLayer.TreeDirChange(Sender: TBaseVirtualTree;
  Node: PVirtualNode);
{
Важнейшая процедура изменения узла дерева
}
var
  CurrentFullPath: string;
Begin
// Получение полного имени папки или файла
  CurrentFullPath := ruTreeDirGetFullPath(Sender, Node);
// Чтение параметров слоя для текущего узла
  ReadLayerParams(CurrentFullPath);
end;

procedure TruTreeLayer.aSendToAcadExecute(Sender: TObject);
{
Важнейшее действие - отправка макроса в AutoCAD
}

begin
  if EditLayerMethod.Text <> '' then
  begin
    AcadApplication.ActiveDocument.Activate;
{ Посылаем в активный документ строку макроса, программно "нажимая" клавишу Enter
в конце.
}
    AcadApplication.ActiveDocument.SendCommand(EditLayerMethod.Text + #13);
  end;
end;

procedure TruTreeLayer.aEditMacroExecute(Sender: TObject);
{
Вызов редактора макроса
}
var
// Форма редактора
  MacroEditor: TMacroEditor;
begin
  MacroEditor := TMacroEditor.Create(Application);
  MacroEditor.EditMacro.Text := EditLayerMethod.Text;
```

```
  if MacroEditor.ShowModal = mrOk then
  EditLayerMethod.Text := MacroEditor.EditMacro.Text;
  MacroEditor.Free;
end;

procedure TruTreeLayer.aAboutExecute(Sender: TObject);
{
Простой вызов COM-сервера для просмотра текстового файла
}
begin
  About.Show(AboutFileName, 'О программе', 'Классификатор слоев',
    False);
end;

procedure TruTreeLayer.aSetLayerInAcadExecute(Sender: TObject);
{
Очень важное действие по установке слоя в AutoCAD
}
var
  xDataType, xDataValue: OleVariant;
  LayerForAcad: string;
  ActiveLayerObj, NewLayerObj:OleVariant;
  ActiveLayerName: string;
  ActiveLayerColor: Integer;
begin
  // Какое имя слоя устанавливаем
  if CheckBoxUseAltName.Checked then LayerForAcad := LayerAltName
  else LayerForAcad := LayerName;
  ActiveLayerObj := AcadApplication.ActiveDocument.ActiveLayer;
  ActiveLayerName := ActiveLayerObj.Name;
// Если заданный слой не является текущим
  if ActiveLayerName <> LayerForAcad then begin
// создаем новый слой и делаем активным
  NewLayerObj := AcadApplication.ActiveDocument.Layers.Add(LayerForAcad);
  AcadApplication.ActiveDocument.ActiveLayer := NewLayerObj;
  ActiveLayerObj := AcadApplication.ActiveDocument.ActiveLayer;
{
 При включенном флажке принудительно восстанавливаем цвет по классификатору
}
  if CheckBoxParamsByClass.Checked then begin
    ActiveLayerColor := ActiveLayerObj.Color;
    if ActiveLayerColor <> LayerColor
      then ActiveLayerObj.Color := LayerColor;
  end;
{
А теперь привязываем к расширенным данным слоя нашу информацию, для того, чтобы сам
слой "знал" о себе дополнительные данные
}
{Регистрируем наше приложение
}
```

```
  AcadApplication.ActiveDocument.RegisteredApplications.Add(ruAppID);
{ Создаем безопасный массив типов и значений данных
}
  xDataType := VarArrayCreate([0, 2], varSmallInt);
  xDataValue := VarArrayCreate([0, 2], varVariant);
{ Первым элементом всегда должно быть имя приложения с кодом 1001
}
  xDataType[0] := 1001;
  xDataValue[0] := ruAppId;
{Далее в виде строки (код 1000) записываем относительное имя INI-файла слоя, чтобы
внутри AutoCAD можно было взять оттуда любые данные для слоя
}
  xDataType[1] := 1000;
  LayerShortIni := ruExcludeRootDir(LayerFileName, LayerDir);
  xDataValue[1] := LayerShortIni;
{ Добавляем номер цвета слоя по классификатору, чтобы после изменений,
произведенных пользователем, можно было установить "стандартный" цвет
}

  xDataType[2] := 1070;
  xDataValue[2] := LayerColor;
{Связывание расширенных данных со слоем}
  ActiveLayerObj.SetXData(xDataType, xDataValue);
 end;
end;

procedure TruTreeLayer.aReloadExecute(Sender: TObject);
{
Перегрузка дерева по любому поводу
}
begin
  LoadDir(TreeRootDir);
end;

function TruTreeLayer.RelativeLayerFileName(LayerFileName: string): string;
{
Вычисление относительного имени слоя
}
begin
  Result := LayerFileName;
  StrReplace(Result, LayerDir, '', [rfReplaceAll, rfIgnoreCase]);
end;

function TruTreeLayer.FileNameToTreePath(FileName, Ext: string): string;
{
Преобразование полного имени слоя в относительный путь в дереве
}
begin
  Result := FileName;
// Удаление корневого каталога
  StrReplace(Result, LayerDir, '', [rfReplaceAll, rfIgnoreCase]);
```

```
// Удаление расширения
  StrReplace(Result, Ext, '', [rfReplaceAll, rfIgnoreCase]);
end;

function TruTreeLayer.TreePathToFileName(TreePath: string): string;
{
Преобразование пути в дереве в имя файла
}
var
  Ext: string;
begin
  Ext := ExtractFileExt(TreePath);
  if Ext = LayerExt then Result := TreePath
  else
  begin
    Result := ruUpDir(TreePath);
    Result := ruIncludeTrailingBackslash(Result) +
     ruExtractFileBaseName(TreePath) + LayerFolderExt;
  end;
end;

function TruTreeLayer.LayerFileNameToLayerName(FileName: string): string;
{
Преобразование имени файла слоя в имя слоя
}
var
  S: string;
begin
{
 Для папки передается имя файла в вышележащем каталоге
 Для самого корня вышележащий каталог не включает уже LayerDir
}
  s := ruExcludeFileExt(FileName);
{
  Удаляем из имени корневой каталог слоев
 }
  s := ruExcludeRootDir(ruIncludeTrailingBackslash(s), LayerDir);
  if s = '' then
  begin
    Result := XmlRootName + RuLayerLabel;
    Exit;
  end;
  Result := ruExcludeTrailingBackslash(s);
// Замена слэшей и пробелов на символы-заменители "_"
  Result := Q_ReplaceStr(Result, PathDividerChar, DividerLayer);
  Result := Q_ReplaceStr(Result, ' ', RuReplacerSpace) + RuLayerLabel;
end;

procedure TruTreeLayer.AcadColorSelChange(Sender: TObject; wParam,
  lParam: Integer);
{
Обработка выбора цвета из раскрывающегося списка
}
```

```
var
  Color: integer;
begin
  Color := AcadColor.GetSelColorIndex;
  if (Color <> LayerColor) then
  begin
    LayerColor := Color;
    WriteIntToLayerIni(LayerFileName, LayerColorVar, LayerColor);
  end;
end;

procedure TruTreeLayer.CheckBoxSpecLayerClick(Sender: TObject);
{
Обработка изменения флажка "Специальный слой"
}
var
  isSpecial: boolean;

begin
  isSpecial := CheckBoxSpecLayer.Checked;
  if isSpecial <> LayerIsSpecial then
  begin
    LayerIsSpecial := IsSpecial;
// Записываем в INI-файл слоя признак "специальности"
    WriteBoolToLayerIni(LayerFileName, LayerIsSpecialVar,
     LayerIsSpecial);
  end;
end;

procedure TruTreeLayer.CheckBoxUseAltNameClick(Sender: TObject);
{
Обработка переключения флажка "Второе имя"
}
var
  AltName: boolean;

begin
  AltName := CheckBoxUseAltName.Checked;
  if AltName <> LayerUseAltName then
  begin
    LayerUseAltName := AltName;
    WriteBoolToLayerIni(LayerFileName, LayerUseAltNameVar,
    LayerUseAltName);
  end;
end;

procedure TruTreeLayer.AddLayerToTree(ChildLayer: boolean);
{
Добавление нового слоя в дерево
}
var
  CurrFileBase,CurrDir,RenamedLayerFileName, CurrLayerFileName,
  CurrExt, NewDir, NewLayer: string;
```

```
  NewNode, CurrNode: PVirtualNode;
  Data : PShellObjectData;
begin
  CurrNode := TreeDir.FocusedNode;
  if Assigned(CurrNode) then
  begin
{
Если добавляем в папку, то надо создать каталог
Если добавляем потомка к обычному узлу, надо переименовать INI обычного в INI папки
}
  CurrLayerFileName:=LayerFileName;
  CurrExt:=ExtractFileExt(CurrLayerFileName);
  CurrDir:= ExtractFilePath(CurrLayerFileName);
  CurrFileBase:=ruExtractFileBaseName(CurrLayerFileName);
  NewLayer := InputNewLayerName;
  if ChildLayer then
    begin
      NewNode := TreeDir.AddChild(CurrNode);
      // Новый каталог надо получить из старого базового имени
      NewDir:=CurrDir+CurrFileBase;
      if not DirectoryExists(NewDir) then  ForceDirectories(NewDir);
      LayerFileName := ruIncludeTrailingBackslash(NewDir) + NewLayer
       + LayerExt;
      RenamedLayerFileName:=ChangeFileExt(CurrLayerFileName,
      LayerFolderExt);
      RenameFile(CurrLayerFileName,RenamedLayerFileName);
      // Надо изменить данные и у родителя
      Data := TreeDir.GetNodeData(CurrNode);
      Data.FullPath := RenamedLayerFileName;
      TreeDir.ReinitNode(CurrNode, False);
    end
    else begin
      NewNode := TreeDir.AddChild(CurrNode.Parent);
      LayerFileName := ruIncludeTrailingBackslash(CurrDir) + NewLayer
      + LayerExt;
    end;
    WriteStringToLayerIni(LayerFileName, LayerNameVar, NewLayer);
    Data := TreeDir.GetNodeData(NewNode);
    Data.FullPath := LayerFileName;
    Data.Attributes := ruDirTreeReadAttributes(LayerFileName);
    TreeDir.ReinitNode(NewNode, False);
  end;
 end;

function TruTreeLayer.InputNewLayerName: string;
{
Ввод имени нового слоя
}

begin
  Inc(NewNumber);
  Result := 'Слой' + IntToStr(NewNumber);
```

```
  if not InputQuery('Создание слоя', 'Введи имя нового слоя:', Result)
  then exit;
  Result := Q_ReplaceStr(Result, PathDividerChar, '-');
  Result := Q_ReplaceStr(Result, ' ', '-');
  Result := Q_ReplaceStr(Result, DividerLayer, '-');
end;

procedure TruTreeLayer.aTreeAddChildExecute(Sender: TObject);
{
Действие добавления потомка в дерево
}
begin
  AddLayerToTree(True);
end;

procedure TruTreeLayer.aTreeAddExecute(Sender: TObject);
{
Действие добавления узла в дерево
}
begin
  AddLayerToTree(False);
end;

procedure TruTreeLayer.aExportToXmlExecute(Sender: TObject);
{
Действие экспорта классификатора в XML-формат
}
begin
  SaveToXML;
end;

procedure AddAttrib(var DomNode: TjanXMLNode2; AttrName, AttrValue: string);
{
Экспорт в XML:
Добавление атрибутов к узлу дерева
}
var
  Attrib: TjanXMLattribute2;
begin
  Attrib := TjanXMLattribute2.Create;
  Attrib.Name := AttrName;
  Attrib.value := AttrValue;
  DomNode.attributes.Add(Attrib);
end;

procedure TruTreeLayer.SaveToXML;
{
Экспорт в XML:
Выполняем действия, знакомые по главе 16 - связываем с узлами дерева узлы модели
документа с последующим сохранением в файл
}
```

```
var
  DOM: TjanXMLParser2;
  CurrDomNode: TjanXMLNode2;
   Dir,FileName: string;
  Data: PShellObjectData;
  CurrTreeNode : PVirtualNode;
begin
    ShowSplash('Ждите, перечитываю весь классификатор',False);
// Обязательно построить все дерево
    TreeDir.FullExpand;
    DOM := TjanXMLparser2.Create;
    DOM.Xml := '<'+XmlRoot+'></'+XmlRoot+'>';
    CurrDomNode := DOM;
    CurrTreeNode :=TreeDir.GetFirst;
    Data:=TreeDir.GetNodeData(CurrTreeNode);
    Dir := Data.FullPath;
    FileName:=ruIncludeTrailingBackslash(Dir)+Data.Display+'.xml';
    AddAttrib(CurrDomNode, 'name', Data.Display);
    SetSplashText('Создаю XML-документ' + #13#10+ FileName);
    GenerateXML(CurrDomNode,CurrTreeNode);
    DOM.SaveXML(FileName);
    DOM.Free;
    HideSplash;
end;

procedure TruTreeLayer.GenerateXML(StartDomNode: TjanXMLNode2; StartTreeNode:
PVirtualNode);
{
Рекурсивное построение модели XML-документа
}
var
  CurrTreeNode : PVirtualNode;
  ChildDomNode: TjanXMLNode2;
begin
  CurrTreeNode := StartTreeNode.FirstChild;
  while (CurrTreeNode <> nil) do
  begin
    ChildDomNode := TjanXMLNode2.Create;
    StartDomNode.AddNode(ChildDomNode);
    MakeDomNodeAttribs(ChildDomNode, CurrTreeNode);
    GenerateXML(ChildDomNode, CurrTreeNode);
    CurrTreeNode := CurrTreeNode.NextSibling;
  end;
end;

procedure TruTreeLayer.MakeDomNodeAttribs(CurrDomNode: TjanXMLNode2;
  CurrTreeNode: PVirtualNode);
{
Построение атрибутов для узла
}
```

```
var
  Text, FileName: string;
  Data: PShellObjectData;

begin
    Data:=TreeDir.GetNodeData(CurrTreeNode);
    Text:=Data.Display;
    Text:=ruExcludeFileExt(Text);
    FileName:=ruTreeDirGetFullPath(TreeDir, CurrTreeNode);
    MakeLayerVars(FileName,True);
    CurrDomNode.name := LayerName;
    AddAttrib(CurrDomNode, 'name', Text);
    AddAttrib(CurrDomNode, LayerAltNameVar, LayerAltName);
    AddAttrib(CurrDomNode, LayerCommentVar, LayerComment);
    AddAttrib(CurrDomNode, LayerColorVar, IntToStr(LayerColor));
    AddAttrib(CurrDomNode, LayerImageVar, LayerImageFile);
    AddAttrib(CurrDomNode, LayerMethodVar, LayerMethod);
    AddAttrib(CurrDomNode, LayerMethodCommentVar, LayerMethodComment);
    AddAttrib(CurrDomNode, LayerUseAltNameVar,
     BoolToStr(LayerUseAltName));
    AddAttrib(CurrDomNode, LayerIsSpecialVar, BoolToStr(LayerIsSpecial));
end;

procedure TruTreeLayer.JvDirectoryEditAfterDialog(Sender: TObject;
  var Name: string; var Action: Boolean);
{
Возможность изменить корневой каталог классификатора
}
var
  RootPos: integer;
begin
  RootPos := Q_PosText(LayerDir, Name, 0);
  if RootPos = 0 then
  begin
    Application.MessageBox(PChar('Нельзя выбирать каталог слоев вне ' +
      LayerDir), 'Ошибка', MB_OK + MB_ICONHAND + MB_DEFBUTTON1 + MB_APPLMODAL);
    JvDirectoryEdit.InitialDir := TreeRootDir;
    JvDirectoryEdit.Text := TreeRootDir;
    Name := TreeRootDir;
    Exit;
  end;
  if DirectoryExists(Name) then LoadDir(Name);
end;

procedure TruTreeLayer.MakeLayerVars(NodePathString: string; DeleteRoot:
  boolean);
{
Чтение параметров слоя из файла
}
var
  ini: TIniFile;
```

```
begin
  LayerFileName := TreePathToFileName(NodePathString);
  LayerName := LayerFileNameToLayerName(LayerFileName);
  StatusBar.SimpleText := LayerFileName;
  ini := TIniFile.Create(LayerFileName);
  LayerIsSpecial := ini.ReadBool(LayerIniSection, LayerIsSpecialVar, False);
  if LayerIsSpecial then
    LayerName := ini.ReadString(LayerIniSection, LayerNameVar, LayerName);
  LayerUseAltName := ini.ReadBool(LayerIniSection, LayerUseAltNameVar, False);
  LayerAltName := ini.ReadString(LayerIniSection, LayerAltNameVar, ruUnicalStr);
  LayerComment := ini.ReadString(LayerIniSection, LayerCommentVar, LayerName);
  LayerColor := ini.ReadInteger(LayerIniSection, LayerColorVar, 7);
  LayerImageFile := ChangeFileExt(LayerFileName, '.jpg');
  if DeleteRoot then StrReplace(LayerImageFile, LayerDir, '',
    [rfReplaceAll, rfIgnoreCase]);
  LayerMethod := ini.ReadString(LayerIniSection, LayerMethodVar, '');
  LayerMethodComment := ini.ReadString(LayerIniSection,
      LayerMethodCommentVar,'');
  Ini.Free;
end;

procedure TruTreeLayer.ReadLayerParams(CurrentFullPath: string);
{
Чтение параметров слоя и их отображение
}
begin
  MakeLayerVars(CurrentFullPath, False);
  CheckBoxSpecLayer.Checked := LayerIsSpecial;
  EditLayerName.Text := LayerName;
  EditLayerName.Button.Enabled := False;
  CheckBoxUseAltName.Checked := LayerUseAltName;
  EditLayerAltName.Text := LayerAltName;
  EditLayerAltName.Button.Enabled := False;
  EditLayerComment.Text := LayerComment;
  AcadColor.SetSelColorIndex(LayerColor);
  Image.Visible := False;
  if FileExists(LayerImageFile) then
    LoadImageFromFile(LayerImageFile, Image);
  EditLayerMethod.Text := LayerMethod;
  EditLayerMethodComment.Text := LayerMethodComment;
end;

procedure TruTreeLayer.aExpandExecute(Sender: TObject);
{
Полное развертывание дерева. Может занять много времени, поэтому выводим
предупреждающее сообщение
}
begin
  if SlowExpanded then
    ShowSplash('Ждите, перечитываю весь классификатор', False);
  TreeDir.FullExpand;
```

```
  SlowExpanded := False;
  HideSplash;
end;

procedure TruTreeLayer.aCollapseExecute(Sender: TObject);
{
Сворачивание дерева
}
begin
  TreeDir.FullCollapse;
end;

procedure TruTreeLayer.EditLayerNameButtonClick(Sender: TObject);
{
Щелчок по кнопке сохранения имени слоя
}
var
  NewLayerName: string;
begin
  NewLayerName := EditLayerName.Text;
  if (NewLayerName <> LayerName) and (NewLayerName <> '') then
  begin
    LayerName := RemoveBadChars(NewLayerName);
    WriteStringToLayerIni(LayerFileName, LayerNameVar, LayerName);
    EditLayerName.Button.Enabled := False;
  end;
end;

procedure TruTreeLayer.EditLayerNameChange(Sender: TObject);
{
Проверка доступности кнопки сохранения
}
begin
  EditLayerName.Button.Enabled := (EditLayerName.Text <> LayerName);
end;

procedure TruTreeLayer.EditLayerAltNameButtonClick(Sender: TObject);
{
Сохранение второго имени слоя
}
var
  S: string;
begin
  S := EditLayerAltName.Text;
  if (S <> LayerAltName) then
  begin
    LayerAltName := RemoveBadChars(s);
    WriteStringToLayerIni(LayerFileName, LayerAltNameVar, LayerAltName);
    EditLayerAltName.Button.Enabled := False;
  end;
end;
```

```
procedure TruTreeLayer.EditLayerAltNameChange(Sender: TObject);
{
Проверка доступности кнопки сохранения
}
begin
  EditLayerAltName.Button.Enabled := (EditLayerAltName.Text <> LayerAltName);
end;

procedure TruTreeLayer.EditLayerCommentButtonClick(Sender: TObject);
{
Сохранение комментария к слою
}
var
  S: string;
begin
  S := EditLayerComment.Text;
  if (S <> LayerComment) then
  begin
    LayerComment := S;
    WriteStringToLayerIni(LayerFileName, LayerCommentVar, LayerComment);
    EditLayerComment.Button.Enabled := False;
  end;
end;

procedure TruTreeLayer.EditLayerCommentChange(Sender: TObject);
{
Проверка доступности кнопки сохранения
}
begin
  EditLayerComment.Button.Enabled := (EditLayerComment.Text <> LayerComment);
end;

procedure TruTreeLayer.EditLayerMethodChange(Sender: TObject);
{
Проверка доступности кнопки сохранения
}
begin
  EditLayerMethod.Button.Enabled := (EditLayerMethod.Text <> LayerMethod);
end;

procedure TruTreeLayer.EditLayerMethodButtonClick(Sender: TObject);
{
Сохранение макроса слоя
}
var
  s: string;
begin
  S := EditLayerMethod.Text;
  if (S <> LayerMethod) and (S <> '') then
  begin
    LayerMethod := S;
    WriteStringToLayerIni(LayerFileName, LayerMethodVar, LayerMethod);
  end;

end;
```

```
procedure TruTreeLayer.EditLayerMethodCommentButtonClick(Sender: TObject);
{
Сохранение комментария к макросу
}
var
  s: string;
begin
  S := EditLayerMethodComment.Text;
  if (S <> LayerMethodComment) and (S <> '') then
  begin
    LayerMethodComment := S;
    WriteStringToLayerIni(LayerFileName, LayerMethodCommentVar,
      LayerMethodComment);
  end;
end;

procedure TruTreeLayer.EditLayerMethodCommentChange(Sender: TObject);
{
Проверка доступности кнопки сохранения
}
begin
  EditLayerMethodComment.Button.Enabled := (EditLayerMethodComment.Text <>
LayerMethodComment);
end;

procedure TruTreeLayer.ActionList1Update(Action: TBasicAction;
  var Handled: Boolean);
{
Проверка доступности визуальных элементов во время пауз
}
begin
  EditLayerName.Enabled := LayerIsSpecial;
  aPasteFromClip.Enabled := not (LayerImageFile = '');
  aPasteStandardImage.Enabled := aPasteFromClip.Enabled;
  aPasteImageFree.Enabled := aPasteFromClip.Enabled;
  aShowFullImage.Enabled := aPasteFromClip.Enabled;
  aTreeAdd.Enabled := (LayerName <> '');
  aEditMacro.Enabled := (LayerName <> '');
  aSendToAcad.Enabled := (LayerMethod <> '') and AcadFound;
  aSetLayerInAcad.Enabled := (LayerName <> '') and AcadFound;
end;

procedure TruTreeLayer.aPasteFromClipExecute(Sender: TObject);
{
Вставка иллюстрации из буфера обмена
}
begin
  PasteFromClipboard(Image, LayerImageFile);
end;

procedure TruTreeLayer.aPasteStandardImageExecute(Sender: TObject);
{
Вставка иллюстрации стандартного размера
}
```

```
begin
  Hide;
  Application.ProcessMessages;
  Sleep(500);
  PasteStandardImage(Image, LayerImageFile, Image.Width, Image.Height);
  show;
end;

procedure TruTreeLayer.aPasteImageFreeExecute(Sender: TObject);
{
Вставка иллюстрации произвольного размера
}
begin
  Hide;
  Application.ProcessMessages;
  Sleep(500);
  PasteFreeSizeImage(Image, LayerImageFile);
  Show;
end;

procedure TruTreeLayer.aShowFullImageExecute(Sender: TObject);
{
Просмотр иллюстрации в натуральную величину
}
begin
  ShowFullXmlImage(LayerImageFile, LayerName);
end;

procedure TruTreeLayer.aExitExecute(Sender: TObject);
{
Выход из программы
}
begin
  Close;
end;
end.
```

Листинг 18.4. Файл ruTreeDirUtils.pas

```
unit ruTreeDirUtils;
{
Модуль процедур и функций для работы с виртуальными деревьями каталогов. Здесь мы
ничего особого не изобретали, просто адаптировали поставляемые с компонентом
примеры к нашим условиям
}
interface
uses
  VirtualTrees;

Type
{Объявление типа данных, привязываемых к узлам дерева}
  PShellObjectData = ^TShellObjectData;
```

```
  TShellObjectData = record
    FullPath, Display: WideString;
    Attributes: Cardinal;
  end;
procedure ruTreeDirInitNode(var Tree: TBaseVirtualTree; ParentNode,
  Node: PVirtualNode; var InitialStates: TVirtualNodeInitStates;
  RootDir: string);
procedure ruTreeDirInitChildren(var Tree: TBaseVirtualTree;
  Node: PVirtualNode; var ChildCount: Cardinal; FileExt: string);
procedure ruTreeDirFreeNode(var Tree: TBaseVirtualTree;
   Node: PVirtualNode);
function ruTreeDirCompareNodes(Tree: TBaseVirtualTree; Node1, Node2:
  PVirtualNode): Integer;
function ruTreeDirGetFullPath(Tree: TBaseVirtualTree;
  Node: PVirtualNode): WideString;
function ruTreeDirGetText(Tree: TBaseVirtualTree; Node: PVirtualNode):
  WideString;
function ruDirTreeReadAttributes(const Name: WideString): Cardinal;
procedure ruTreeDirInit(var TreeDir: TVirtualStringTree);

implementation

uses
  Controls, Forms, ShlObj, ActiveX, ruFileUtils, SysUtils;

procedure ruTreeDirInitNode(var Tree: TBaseVirtualTree; ParentNode,
  Node: PVirtualNode; var InitialStates: TVirtualNodeInitStates; RootDir:
    string);
{Инициализация узла дерева}
var
  Data: PShellObjectData;
begin
  Data := Tree.GetNodeData(Node);
  if ParentNode = nil then
  begin
    Data.FullPath := RootDir;
    Data.Display :=
    ExtractFileName(ruExcludeTrailingBackslash(Data.FullPath));
  end else begin
    Data.Display :=
    ExtractFileName(ruExcludeTrailingBackslash(Data.FullPath));
    if (Data.Attributes and SFGAO_FOLDER) = 0 then
  end;
  Data.Attributes := ruDirTreeReadAttributes(Data.FullPath);
  if ((Data.Attributes and SFGAO_HASSUBFOLDER) <> 0) or
    (((Data.Attributes and SFGAO_FOLDER) <> 0) and
    ruFolderHasChildren(Data.FullPath)) then
    Include(InitialStates, ivsHasChildren);
end;

procedure ruTreeDirInitChildren(var Tree: TBaseVirtualTree;
  Node: PVirtualNode; var ChildCount: Cardinal; FileExt: string);
```

```
{Инициализация узла-потомка}
var
  Data, ChildData: PShellObjectData;
  SR: TSearchRec;
  ChildNode: PVirtualNode;
  NewName: string;

begin
  Data := Tree.GetNodeData(Node);
  if FindFirst(ruIncludeTrailingBackslash(Data.FullPath) + '*.*',
  faAnyFile, SR) = 0 then
  begin
    Screen.Cursor := crHourGlass;
    try
      repeat
        if (SR.Name <> '.') and (SR.Name <> '..') then
        begin
          NewName := ruIncludeTrailingBackslash(Data.FullPath) + SR.Name;
          if (SR.Attr and faDirectory <> 0) or ruFileExtValid(NewName,
           FileExt)
            then begin
            ChildNode := Tree.AddChild(Node);
            ChildData := Tree.GetNodeData(ChildNode);
            ChildData.FullPath := NewName;
            ChildData.Attributes := ruDirTreeReadAttributes(NewName);
            Tree.ValidateNode(Node, False);
          end;
        end;
      until FindNext(SR) <> 0;
      ChildCount := Tree.ChildCount[Node];
      if ChildCount > 0 then
        Tree.Sort(Node, 0, TVirtualStringTree(Tree).Header.SortDirection,
          False);
    finally
      FindClose(SR);
      Screen.Cursor := crDefault;
    end;
  end;
end;

procedure ruTreeDirFreeNode(var Tree: TBaseVirtualTree;
           Node: PVirtualNode);
var
  Data: PShellObjectData;
begin
  Data := Tree.GetNodeData(Node);
  Finalize(Data^); // Clear string data.
end;

function ruTreeDirCompareNodes(Tree: TBaseVirtualTree; Node1, Node2:
  PVirtualNode): Integer;
```

```
{ Функция сравнения двух узлов. Используется для того, чтобы в дереве папки
отображались выше файлов}
var
  Data1, Data2: PShellObjectData;
begin
  Data1 := Tree.GetNodeData(Node1);
  Data2 := Tree.GetNodeData(Node2);
  if ((Data1.Attributes xor Data2.Attributes) and SFGAO_FOLDER) <> 0 then
  begin
   if (Data1.Attributes and SFGAO_FOLDER) <> 0 then Result := -1
    else  Result := 1;
  end else
   Result := CompareText(Data1.FullPath, Data2.FullPath);
end;

function ruTreeDirGetFullPath(Tree: TBaseVirtualTree;
                Node: PVirtualNode): WideString;
{Получение полного имени папки или файла в узле дерева}
var
  Data: PShellObjectData;
begin
  Result := '';
  Data := Tree.GetNodeData(Node);
  if Assigned(Data) then Result := Data.FullPath;
end;

function ruTreeDirGetText(Tree: TBaseVirtualTree; Node: PVirtualNode):
  WideString;
{Получение текста, отображаемого в узле дерева}
var
  Data: PShellObjectData;
begin
  Result := '';
  Data := Tree.GetNodeData(Node);
{ Мы не даем показывать расширение, чтобы пользователь и не знал, что работает
с файловой системой, а не с базой данных}
  if Assigned(Data) then Result := ruExcludeFileExt(Data.Display);
end;

function ruDirTreeReadAttributes(const Name: WideString): Cardinal;
{Получение атрибутов файла или папки}

const
  SFGAO_CONTENTSMASK = $F0000000;

var
  Desktop: IShellFolder;
  Eaten: Cardinal;
  PIDL: PItemIDList;
  Malloc: IMalloc;

begin
  // Корневая папка пространства имен Shell всегда Desktop.
  SHGetDesktopFolder(Desktop);
```

```
  Result := SFGAO_DISPLAYATTRMASK or SFGAO_CONTENTSMASK or SFGAO_COMPRESSED;
  Desktop.ParseDisplayName(0, nil, PWideChar(Name), Eaten, PIDL, Result);
  SHGetMalloc(Malloc);
  Malloc.Free(PIDL);
end;

procedure ruTreeDirInit(var TreeDir: TVirtualStringTree);
{Инициализация дерева. Всего лишь выделение памяти для данных}
begin
  TreeDir.NodeDataSize := SizeOf(TShellObjectData);
  TreeDir.RootNodeCount := 1;
end;
end.
```

В тексте программы был предусмотрен экспорт классификатора в XML-формат. Фрагмент получаемого файла приведен в листинге 18.1. Нетрудно выполнить и экспорт в базу данных. Самым рациональным было бы создание SQL-скрипта, выполнив который можно сразу создать и заполнить данными базу данных в формате, например, Interbase.

Запуск программы-классификатора

Для облегчения запуска программы мы предусмотрим два варианта.

Во-первых, включим в XML-меню приложений программы-стартера *(см. главу 20)* строку для запуска программы-классификатора:

```
<app name='Классификатор' dir='%BIN%' exe='ruLayersExplorer'
paramstr=''>Классификатор слоев</app>
```

Во-вторых, включим в меню ruCAD.mnu в подходящих местах макрос для запуска классификатора, например, к одной из кнопок панели инструментов:

```
RU_LAY_CLASS_SEL [_Button("Запуск классификатора слоев", RU_LAY_CLASS_SEL_16,
RU_LAY_CLASS_SEL_24)]^C^C^P(ru-app-load "ru_layer_class_select")
```

Текст программы запуска классификатора представлен в листинге 18.5.

Листинг 18.5. Файл ru_layer_class_select.lsp

```
(defun START (/)
 (ru-app-run (ru-file-bin "ruLayersExplorer.exe") false)
(princ "\nЗапущен классификатор слоев")
(princ)
)
(START)
```

Формирование библиотеки функций для работы с классификатором

Теперь разработаем несколько LISP-функций для работы со слоями с учетом возможностей, предоставляемых классификатором. Напоминаем, что при установке

классифицированного слоя создаются расширенные данные для слоя (см. листинг 18.5):

- код 1001 — имя нашего приложения;
- код 1000 — относительное имя INI-файла слоя;
- код 1070 — номер цвета слоя.

Нам нужно извлечь эти данные и использовать по назначению. Расширенные данные слоя можно получить с помощью функции, приведенной в листинге 18.6.

Листинг 18.6. Функция `ru-xdata-get-ruclass-for-layer-obj`

```
(defun ru-xdata-get-ruclass-for-layer-obj (layer_obj / types values layer_name)
;;; Получает расширенные данные ruCAD для классифицированного слоя
;;; Аргумент: VLA-объект слоя
  (vla-getxdata layer_obj '"RUCAD" 'types 'values)
  (list (ru-obj-get-obj-name layer_obj)
   (ru-list-clear-elem (ru-obj-make-list-lisp-types-values types values)
   (cons 1001 "RUCAD") nil))
)
```

Получив расширенные данные одного слоя, легко получить их и для всех слоев рисунка (листинг 18.7).

Листинг 18.7. Функция `ru-xdata-get-ruclass-for-all-layers`

```
(defun ru-xdata-get-ruclass-for-all-layers (/ xdata result)
;;; расширенные данные ruCAD из классификатора для всех слоев
;| Пример:
(ru-xdata-get-ruclass-for-all-layers)
(("0" nil)
   ("P_Общие_Заглавные_Таблицы_ru"
     ((1000 . "P\\Общие\\Заглавные\\Таблицы.lay") (1070 . 7)))
   ("P_Общие_Заглавные_Указания_ru"
     ((1000 . "P\\Общие\\Заглавные\\Указания.lay") (1070 . 7)))
    ("P_Общие_Заглавные_Формат_ru"
     ((1000 . "P\\Общие\\Заглавные\\Формат.lay") (1070 . 7)))
    ("Топо_Водоемы_Болота_ru"
     ((1000 . "Топо\\Водоемы\\Болота.layf") (1070 . 7)))
)
|;
  (vlax-for each (ru-obj-get-layers)
    (setq xdata (ru-xdata-get-ruclass-for-layer-obj each)
     result (cons xdata result))
 )
  (reverse result)
)
```

Наверняка нам часто понадобятся данные для текущего слоя (листинг 18.8).

Листинг 18.8. Функция `ru-xdata-get-ruclass-for-active-layer`

```
(defun ru-xdata-get-ruclass-for-active-layer ()
;;; расширенные данные ruCAD из классификатора для текущего слоя
;|Пример:
(ru-xdata-get-ruclass-for-active-layer)
((1000 . "\\Спец\\ОВ\\Отопление\\Оси.lay") (1070 . 7))
|;
  (ru-xdata-get-ruclass-for-layer-obj
    (vla-get-activelayer (ru-obj-get-active-document))
 )
)
```

Практическое применение имеет функция получения данных по имени слоя (листинг 18.9).

Листинг 18.9. Функция `ru-xdata-get-ruclass-by-layer-name`

```
(defun ru-xdata-get-ruclass-by-layer-name (layer_name / obj_layer)
;;; Свойства слоя, заданного по имени
(if (setq obj_layer (ru-obj-layer-by-name layer_name))
    (ru-xdata-get-ruclass-for-layer-obj obj_layer) nil)
)
```

Для дополнительной информации о слое, не "зашитой" в рисунке, необходимо знать имя INI-файла слоя (листинг 18.10).

Листинг 18.10. Функция `ru-file-layer-ini`

```
(defun ru-file-layer-ini (layer_name / xdata ini_file)
;;; Получение имени INI-файла слоя, заданного по имени
;| Пример:
(ru-file-layer-ini "P_Общие_Заглавные_Указания_ru")
"c:\\.ru\\cad\\All Users\\Layers\\Все\\P\\Общие\\Заглавные\\Указания.lay"
(ru-file-layer-ini "0") -> NIL
|;
  (if (setq xdata (ru-xdata-get-ruclass-by-layer-name layer_name))
    (if (setq ini_file (car (ru-list-massoc 1000 (cadr xdata))))
      (strcat (ru-dirs-get-layers-class) ini_file)
      nil
   );_ end of if
    nil
 );_ end of if
)
```

Зная имя INI-файла слоя, мы можем прочитать любую переменную слоя (листинг 18.11).

Листинг 18.11. Функция ru-layer-read-var

```
(defun ru-layer-read-var (layer_name var_name default / ini_name)
 ;;Чтение переменной из INI-файла слоя
  (if (setq ini_name (ru-file-layer-ini layer_name))
    (ru-ini-read ini_name "LayerParams" var_name default)
    default
 );_ end of if
);_ end of defun
```

Получение списка комментариев ко всем слоям рисунка (листинг 18.12).

Листинг 18.12. Функция ru-layer-read-all-comments

```
(defun ru-layer-read-all-comments (/ result)
  (foreach layer_name (ru-obj-list-layers)
    (setq result (cons (ru-layer-read-var layer_name "LayerComment" "")
      result))
 )
  (reverse result)
);_ end of defun
```

Теперь мы можем разработать диалоговую функцию, позволяющую произвести выбор из списка комментированных слоев (листинг 18.13).

Листинг 18.13. Функция ru-layer-show-list-names-comments

```
(defun ru-layer-show-list-names-comments (header)
;;; Пример: (ru-layer-show-list-names-comments "Выбор слоя")
;;; Возвращает список ("Слой" "Комментарий") или nil
  (ru-dlg-double-list header "Слой" "Комментарий" (ru-obj-list-layers)
    (ru-layer-read-all-comments) T)
)
```

Пригодится и выбор слоя указанием объекта в рисунке (листинг 18.14).

Листинг 18.14. Функция ru-get-layer-from-dwg

```
(defun ru-get-layer-from-dwg (message / ent)
;;; Пример: (ru-get-layer-from-dwg "Укажи объект на скрываемом слое")
  (if (setq ent (_ru-get-with-default message "Выход"
       '_ru-get-nentsel-no-error nil nil nil))
      (cdr (assoc 8 (entget (car ent))))
 );_ end of if
)
```

А теперь можно сделать универсальную функцию для ввода имени слоя (листинг 18.15). В прикладных программах она будет применяться очень часто.

Листинг 18.15. Функция ru-get-layer-name

```
(defun ru-get-layer-name (message can_create_new / ent keywords kw_lst kw_curr
kw_new new_layer)
;;; Ввод имени слоя
 ;| Пример:
 (ru-get-layer-name "Выбор  слоя" T)
|;
  (setq kw_lst  "Список" kw_curr "Текущий" kw_new "Новый")
  (if can_create_new
    (setq keywords (strcat kw_lst " " kw_curr " " kw_new))
    (setq keywords (strcat kw_lst " " kw_curr))
 );_ end of if
  (if (setq ent (_ru-get-with-default message "Выход"
  '_ru-get-nentsel-no-error nil keywords nil))
    (cond
      ((= ent kw_lst)
;; Выбор слоя из списка
       (car (ru-layer-show-list-names-comments message))
     )
      ((= ent kw_curr)
;; Текущий слой
       (getvar "CLAYER")
     )
      ((= ent kw_new)
;; Запрос имени нового слоя
       (setq new_layer "")
       (while (not (snvalid new_layer))
        (setq new_layer (ru-get-string-reguired "Имя нового слоя"
         "Новый_слой"))
      );_ end of while
       new_layer
     )
      (t
       (cdr (assoc 8 (entget (car ent))))
     )
   );_ end of cond
 );_ end of if
)
```

Вот пример простой программы, выводящей справку по слою (листинг 18.6).

Листинг 18.16. Файл ru_layer_comment.lsp

```
(defun START (/ layer)
;;;вывод справки по слою
  (while (setq layer (ru-get-layer-name "Выбор слоя для справки" nil))
     (ru-layer-show-info layer)
 );_ end of while
  (princ)
);_ end of defun
(START)
```

ГЛАВА 19

Работа с базами данных

Сначала определимся с терминологией — что же такое *"база данных"*. На этот прямой вопрос Яндекс дает ответ: "Результат поиска: страниц — 1 202 364, серверов — не менее 4 536". Все страницы мы, конечно, не просмотрели, но результаты удивляют. В большинстве случаев это рассуждения "вокруг да около" или "это когда...".

В соответствии с "Законом РФ о правовой охране программ для ЭВМ и баз данных" "база данных — это объективная форма представления и организации совокупности данных (например: статей, расчетов), систематизированных таким образом, чтобы эти данные могли быть найдены и обработаны с помощью ЭВМ". Это очень верное юридическое определение, но нас оно пока не интересует. Пока не придется дискутировать по этим вопросам с прокурором.

Часто встречается определение: "база данных — это набор информации, организованной тем или иным способом". Такая расплывчатая формулировка позволяет объявить базой данных почти все, чем и пользуются некоторые авторы, рассматривающие в качестве работы с базами данных в AutoLISP обычные манипуляции с ассоциированными списками, даже не рассматривая такие частности, как сохранение информации на магнитный носитель.

Безусловные авторитеты в области баз данных (например, Мартин Грубер) выражаются более конкретно, но непонятно: "*реляционная база данных* — это тело связанной информации, сохраняемой в двумерных таблицах", и далее разъясняют это на нескольких сотнях страниц.

Популяризаторы легко манипулируют терминами: "На самом деле, базы данных — штука довольно простая. В принципе, это *всего лишь* программа, обеспечивающая работу с какими-то вашими данными. Все, что база умеет, — это читать данные из файла (и писать в него), сортировать их, выбирать по какому-то критерию". Лихо! И зачем К. Дж. Дейт *всего лишь* "Введение в системы баз данных" написал на 1 070 страницах?

Еще лет 10 назад большинство отечественных пользователей считали, что базы данных — это файлы данных[1], обрабатываемые в таких системах, как FoxPro, Paradox,

[1] Удивительно, но, просмотрев несколько книг по программированию в FoxPro, Clipper, зачитанных нами когда-то до дыр, мы теперь не нашли в них определения термина "база данных". Про "кортежи" и "сущности" написано, а потом сразу — "каждая БД — это файл с расширением".

Clipper, dBASE или Clarion. С переходом на современные многопользвательские системы упрощенные представления о базах данных отошли в прошлое. Хотя, в широком смысле, базой данных можно считать и текстовый файл с *организованной по определенным правилам* информацией, и LISP-список, сохраненный в виде текстового файла, *в контексте этой главы* базой данных (БД) мы будем считать информацию, находящуюся в файлах, создаваемых и обрабатываемых какими-то *системами управления базами данных* (СУБД).

В зависимости от вида организации данных различают несколько моделей представления информации в БД:

- иерархическую;
- сетевую;
- реляционную;
- объектно-ориентированную.

Большинство современных БД для персональных компьютеров являются *реляционными*, т. е. состоящими из совокупности двухмерных таблиц, связанных отношениями.

Систем управления базами данных (СУБД) существует множество. СУБД является комплексом языковых и программных средств, предназначенных для создания и ведения БД. По характеру использования СУБД могут быть *персональными* (Paradox, Access и др.) и *многопользовательскими* (InterBase, Microsoft SQL Server, Oracle, Informix, SyBase и др.), позволяющими создавать информационные системы в архитектуре "клиент-сервер".

В САПР используются мизерные (для СУБД) объемы информации и, чаще всего, можно применить "плоские", т. е. не связанные отношениями таблицы. О некоторых вариантах решения таких задач мы уже писали в *главе 16*.

Небольшие объемы данных, используемых в LISP-программах, и отсутствие, до недавних пор, механизмов связи с "настоящими" базами данных способствовали тому, что разработчики хранили данные во множестве "личных" форматов. Конечно, многие данные можно успешно хранить в текстовом формате, например в виде LISP-списка, который очень удобно мгновенно прочитать из LISP, но очень неудобно разбираться с ним из программы, написанной на любом другом языке. Можно данные хранить и в формате с фиксированной шириной полей или с отделением полей символом-разделителем. Такие данные можно читать на любом языке программирования. Но при использовании и этого, и других "подходящих" форматов придется много заниматься ручным индивидуальным программированием.

При работе с любыми данными их требуется просматривать, редактировать, выбирать требуемые значения. При большом объеме данных необходимы поиск и фильтрация по заданным условиям. Для САПР это не столь актуально, как для других информационных систем, небольшие объемы можно и просто пролистать, но разрабатывать все придется индивидуально. Особенно нудно это делать на таком "невизуальном" языке, как Visual LISP. Если же мы используем технологии баз данных и соответствующие системы разработки, то все делается очень легко.

Например, при использовании Delphi достаточно разместить на форме компоненты `Table`, `DataSource`, `DbGrid` и `DbNavigator`, правильно установить их свойства и, не написав ни одной строчки кода, получить программу, уже умеющую редактировать таблицу базы данных. Аналогичные возможности имеются и в системах разработки

от Microsoft. Конечно, реальные программы работы с базами данных могут быть очень сложными, но усилия программистов в них направлены на отработку логики работы приложения, а не на чтение и отображение данных.

Немного об ADO

Из множества разработанных Microsoft стратегий доступа к данным[1] мы воспользуемся технологией *объектов данных ActiveX* (ActiveX Data Objects) — ADO. Эта технология для нас привлекательна тем, что мы сможем работать с базами данных из Visual LISP. В операционных системах Windows 98 и Windows NT для работы приложений, основанных на ADO, необходимо установить пакет Microsoft Data Access Components (MDAC), в Windows 2000 и Windows XP дополнительный пакет устанавливать не нужно. Обычно все необходимые компоненты ADO устанавливаются с продуктами Microsoft, но можно и скачать пакет MDAC отдельно. На момент написания этих строк последней версией является 2.8. Ссылка на русский комплект MDAC 2.8 появилась 19 сентября 2003 года по адресу:

download.microsoft.com/download/c/f/2/cf2b5cd9-7ffd-4c19-971f-9ccaf0b57d48/MDAC_TYP.EXE

Английскую версию можно закачать с адреса:

download.microsoft.com/download/c/d/f/cdfd58f1-3973-4c51-8851-49ae3777586f/MDAC_TYP.EXE

Используя ADO, программист может получить доступ к любым базам данных. На физическом уровне с базой данных работает приложение-провайдер (provider) через механизмы OLE DB, а ADO предоставляет более удобный интерфейс. Провайдеры для "дружественных" Microsoft продуктов поставляются с MDAC, для технологий фирм-конкурентов, например Borland, провайдера нужно "доставать". *Хороший и бесплатный* провайдер OLE DB, например, для СУБД InterBase найти затруднительно[2]. Если для работы из Delphi с замечательным сервером InterBase и его еще более замечательными бесплатными клонами Firebird и Yaffild технологию ADO применять не обязательно, то для доступа к InterBase из Visual LISP провайдер OLE DB необходим.

Работа с базами данных из Visual LISP

Благодаря поддержке в Visual LISP технологии ActiveX мы можем работать с базами данных из LISP-программ. Конечно, никаких средств для визуализации данных мы при этом не имеем — для этого нужно воспользоваться услугами дополнительных посредников в виде COM-серверов или заниматься нудной работой по отображению данных средствами DCL.

При разработке приложений для работы с базами данных на Delphi можно использовать компоненты для работы с ADO. Эти компоненты позволяют программистам

---

[1] UDA, ODBC, RDO, Jet, DAO, ODBCdirect, OLE DB, RDS, MDAC — как же любит Microsoft запутать пользователей аббревиатурами!

[2] Самый лучший, но платный провайдер см. на **ibprovider.com/rus/**.

использовать привычную логику разработки приложений для БД, усвоенную при работе с "движком" баз данных *BDE* (Borland Database Engine)[1]. Напрямую технологию ADO в этом случае не используют, т. к. она спрятана "оберткой" из компонентов `ADOConnection` и `ADOTable`. Мы могли бы работать с ADO и без размещения компонентов на форме. Для этого нам пришлось бы:

- импортировать библиотеку типов, получить файл ADODB_TLB.pas и включить его в список используемых модулей;
- ввести объявления переменных:
  - `FConnection: _Connection;` и
  - `FRecordSet: _RecordSet;`
- открывать базу данных и набор записей примерно так:

```
FConnection := CoConnection.Create;
FConnection.Open(ConnectionString, 'admin', '', -1);
FRecordSet := CoRecordSet.Create;
FRecordSet.Open('SELECT * FROM ИмяТаблицы ORDER BY ИмяПоля', FConnection,
  adOpenKeySet, adLockOptimistic, adCmdText);
```

- работать с набором данных через свойства и методы этих объектов.

В Visual LISP у нас нет возможности использовать труд авторов компонентов и мы должны разобраться с технологией ADO даже лучше, чем "дельфины". Возможности по визуальному отображению данных у нас ограниченные, а вот управлять данными "втихую" мы можем.

В AutoCAD имеется пример работы с базами данных с использованием технологии ADO (файл \Sample\Database Connectivity\CAO\caotest.lsp), этот пример поможет нам разобраться с проблемой, но может и запутать, т. к. в нем слишком много места занимают попытки отобразить данные средствами DCL.

Подробности технологии ADO

ADO основана на COM-технологии. Все объекты и интерфейсы ADO являются интерфейсами и объектами COM, следовательно, мы можем получить к ним доступ из Visual LISP. ADO является более дружественной оболочкой базовой технологии OLE DB. Мы уже писали, что, имея подходящий провайдер OLE DB, через ADO можно работать с любыми базами данных. Для работы с базами данных Access используется провайдер Jet — механизм, обеспечивающий низкоуровневый доступ к БД формата Access.

Основными объектами ADO являются:

- `Connection` — объект, используемый для соединения с хранилищем данных;
- `RecordSet` — набор данных, получаемых из хранилища;
- `Command` — объект, используемый для выполнения операторов SQL, не возвращающих наборы данных.

[1] Фирма Borland официально прекратила дальнейшее развитие BDE.

Основным свойством объекта `Connection` является `ConnectionString` — текстовая строка, определяющая совокупность параметров соединения с базой данных. Параметры в `ConnectionString` разделяются точкой с запятой. Строку соединения можно "сочинить" (зная, что в нее записывать) или сконструировать с помощью Мастера, вызываемого при щелчке по кнопке в свойстве инспектора объектов Delphi. Подобный Мастер вызывается многими приложениями (рис. 19.1—19.4).

После завершения диалога строка соединения может иметь такой вид:

```
ConnectionString =
 'Provider=Microsoft.Jet.OLEDB.4.0;'+
 'Password="";'+
 'User ID=Admin;'+
 'DataSource='+
 'C:\.ru\cad\Local Settings\Application Data\ruCAD\ru_users.mdb;'+
 'Mode=Share Deny None;Extended Properties="";Jet OLEDB:System'+
 'database="";Jet OLEDB:Registry Path="";Jet OLEDB:Database'+
 ' Password="";Jet OLEDB:Engine Type=4;'+
 'Jet OLEDB:Database Locking Mode=0;'+
 'Jet OLEDB:Global Partial Bulk Ops=2;'+
 'Jet OLEDB:Global Bulk Transactions=1;'+
 'Jet OLEDB:New Database Password="";'+
 ' Jet OLEDB:Create System Database=False;'+
 'Jet OLEDB:Encrypt Database=False;'+
 ' Jet OLEDB:Don'#39't Copy Locale on Compact=False;'+
 ' Jet OLEDB:Compact Without Replica Repair=False;'+
 'Jet OLEDB:SFP=False'
```

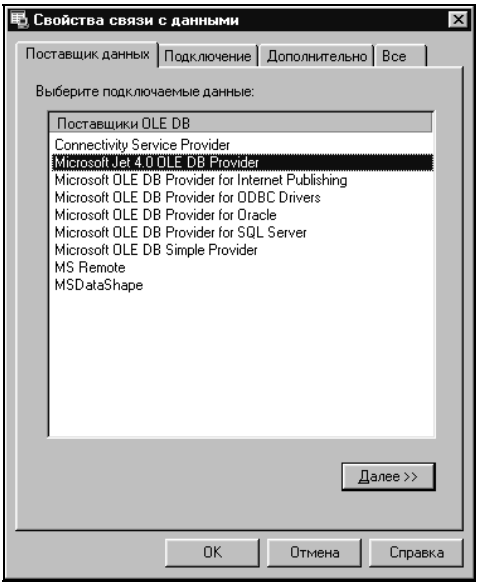


Рис. 19.1. Выбор поставщика данных (провайдера)

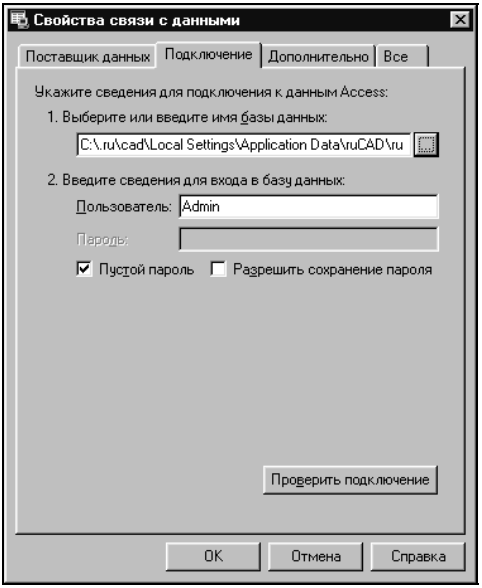


Рис. 19.2. Выбор базы данных и параметров подключения

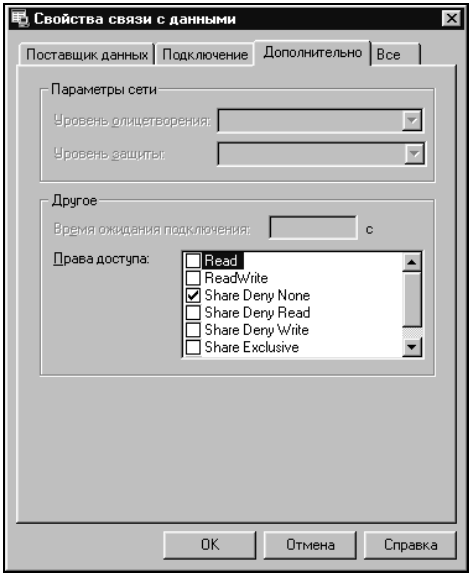


Рис. 19.3. Установка дополнительных условий

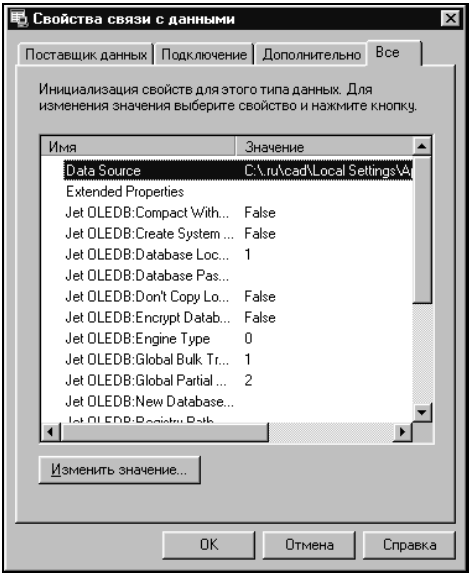


Рис. 19.4. Просмотр и редактирование всех условий

Ничего себе, строчка! Однако уверяем читателей, что мы только выбрали выделенные полужирным параметры — провайдера и файл нашей базы данных. Все остальное является значениями по умолчанию для выбранного провайдера. Разумеется, некоторые значения мы могли бы изменить. Обычно изменяется пароль доступа и режим доступа `Mode`.

Возможен альтернативный вариант `ConnectionString` — ссылка на UDL-файл связи с данными. Выбрать альтернативу в виде UDL-файла можно в редакторе строки соединения. Создать UDL-файл можно в Проводнике Windows, перейдя в нем в нужный каталог, вызвав контекстное меню и выбрав в нем **New | Microsoft Data Link** (Новый | Связь с данными Microsoft)[1]. При создании UDL-файла вызывается Мастер, но строка соединения сохраняется в файле (листинг 19.1).

Листинг 19.1. Файл ru_users.udl

```
[oledb]
; Everything after this line is an OLE DB initstring
Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\.ru\cad\Local Settings
\Application Data\ruCAD\ru_users.mdb;Persist Security Info=False
```

UDL-файл имеет явное преимущество. Его можно редактировать, изменить путь к базе данных и другие параметры. Можно даже изменить провайдера, если он окажется "глючным". Однако параметр `Data Source` содержит конкретное имя файла,

[1] Часто такого пункта в контекстном меню папки не обнаруживается. Чтобы не ставить пользователей в тупик, мы снабдим ruCAD собственным конструктором строки соединения.

привязанное к каталогу C:\.ru\cad\Local Settings\Application Data\ruCAD, а при инсталляции система должна оказаться совсем в другом месте. Следовательно, наша будущая программа-инсталлятор должна не забыть изменить и этот файл.

Замечание

Внешне кажется, что он в формате INI, но хитрость в том, что UDL-файл записан в кодировке Unicode, и обычными способами его безошибочно прочитать не удастся, даже не каждый текстовый редактор его правильно откроет. Программное создание и чтение UDL-файлов необходимо выполнять с помощью специальных функций. Примеры будут приведены ниже.

Нужно ли импортировать библиотеку типов

Практически во всех примерах соединения из Visual LISP с COM-серверами встречается код загрузки библиотеки типов. В CaoTest.lsp это выглядит так:

```
(setq gADO-DLLpath
  "c:\\program files\\common files\\system\\ado\\msado15.dll")
(if (null (findfile gADO-DLLpath))
 (progn
 (alert (strcat "Could not load ADO type library from: " gADO-DLLpath))
 (exit)
)
)
(if (null adok-adStateOpen)
 (vlax-import-type-library :tlb-filename gADO-DLLpath
  :methods-prefix "adom-"
  :properties-prefix "adop-"
  :constants-prefix "adok-"
)
)
```

Конечно, ужасно, что библиотека разыскивается по абсолютному пути — совсем не факт, что на любом компьютере каталог "c:\\Program Files\\Common Files\\" именуется именно так (правильнее обращаться к системной функции `GetCommonFilesFolder`), но не это главное. Основной вопрос — зачем вообще это делается? Мы уже разработали десяток COM-серверов, ни разу не импортировали собственную библиотеку типов, и все прекрасно работает. В то же время мы импортировали в Delphi другие библиотеки типов, например нашей любимой системы AutoCAD. Но Delphi очень умная среда программирования. Если файл xxx_TLB.pas включен в список используемых модулей, то Delphi знает о свойствах и методах объектов этого модуля, компилятор не пропустит неизвестный метод, а программа *Code Insight* (Знаток кода) выдает подсказки об объектах.

В Visual LISP, к сожалению, ничего подобного нет. Единственное удобство заключается в том, что после выполнения функции `vlax-import-type-library` редактор начинает подсвечивать имена переменных и функций, начинающихся с префиксов, заданных с помощью параметров `:methods-prefix`, `:properties-prefix` и `:constants-prefix`. Это хорошо, но не столь важно.

Гораздо важнее другое — во многих библиотеках используются именованные константы. Имеются они в самом языке Visual LISP, например, `vlax-vbEmpty`=0,

vlax-vbNull=1, vlax-vbInteger=2, а vlax-vbString=8. Множество именованных констант имеется и в системе AutoCAD. В своих приложениях мы пока не применяли именованные константы (просто не было необходимости), но в библиотеке ADO они имеются.

Из множества констант и перечислений ADO нам доведется использовать:

- adConnectUnspecified = $FFFFFFFF — обычное значение флага режима для метода Connection.Open;
- adOpenKeyset = $00000001 — одно из возможных значений типа курсора (не экранного, а в контексте терминологии БД) для метода RecordSet.Open. Изменения и удаления, выполненные другими пользователями в наборе записей, будут видны, а добавления не видны;
- adLockOptimistic = $00000003 — одно из допустимых значений типа блокировки для метода RecordSet.Open. Провайдер блокирует запись только при вызове метода Update;
- adCmdText = $00000001 — одно из возможных значений типа команды для метода RecordSet.Open. Команда является текстом (как правило, это выражение языка SQL);
- adStateClosed = $00000000 — одно из возможных состояний набора данных (RecordSet закрыт);
- adStateOpen = $00000001 — одно из возможных состояний набора данных (RecordSet открыт);
- adGetRowsRest = $FFFFFFFF — единственно возможное значение для метода RecordSet.GetRows;
- adSchemaColumns = $00000004 — одно из возможных значений для получения списка таблиц объекта Connection.

Для того чтобы использовать в своих функциях эти и другие константы, мы и должны импортировать библиотеку типов.

Функции для работы с ADO

Формирование библиотеки начнем с функции загрузки библиотеки типов (листинг 19.2).

Листинг 19.2. Функция ru-ado-import-tlb

```
(defun ru-ado-import-tlb (/ ado_clsid ado_dll_file_name)
;;; Проверяем, не загружена ли ранее библиотека типов
(if (null ru-ado-method-append)
  (progn
;;; Проверяем, зарегистрирована ли вообще библиотека ADO
    (if (setq ado_clsid
         (vl-registry-read "HKEY_CLASSES_ROOT\\ADODB.Command\\CLSID")
       );_ end of setq
       (setq ado_dll_file_name (vl-registry-read
            (strcat "HKEY_CLASSES_ROOT\\CLSID\\" ADO_clsid
             "\\InProcServer32")))
```

```
     );_ end of if
;;; Если "прописка" в реестре не найдена, пытаемся найти там, где
;;; библиотека должна находиться теоретически
    (if (not ado_dll_file_name)
        (setq ado_dll_file_name (strcat (ru-dirs-program-files-common)
                        "system\\ado\\msado15.dll"))
   );_ end of if
   (if (findfile ado_dll_file_name)
        (progn
;;; Если библиотека изловлена, импортируем
          (vlax-import-type-library
            :tlb-filename
            ado_dll_file_name
            :methods-prefix
            "ru-ado-method-"
            :properties-prefix
            "ru-ado-prop-"
            :constants-prefix
            "ru-ado-const-"
         );_ end of vlax-import-type-library
       );_ end of progn
        (progn
          (alert (strcat "Не могу найти \n '"
                         ado_dll_file_name
                         "'\nРабота невозможна!"
                 );_ end of strcat
         );_ end of alert
          nil
       );_ end of progn
     );_ end of if
   );_ end of progn
    T
 );_ end of if
)
(ru-ado-import-tlb)
```

Обратите внимание, мы сразу вызываем определенную функцию. Так как все функции будут загружаться в начале работы, то произойдет и автоматический импорт библиотеки типов.

Далее можно переходить к созданию объекта соединения с базой данных (листинг 19.3). Пока будем исходить из предположения, что строка соединения нам известна.

Листинг 19.3. Функция `ru-ado-connect-to-db`

```
(defun ru-ado-connect-to-db (connect_string user_name password /
 connection_object connection_parsing_property connection_properties
                           result temp_object)
;;; Создаем объект соединения с базой данных
```

```
  (setq connection_object (vlax-create-object "ADODB.Connection"))
;;; Пытаемся открыть базу данных, отлавливая возможные ошибки
  (if (vl-catch-all-error-p
        (setq temp_object
               (vl-catch-all-apply
                 'vlax-invoke-method
                 (list connection_object
                       "Open"
                       connect_string
                       user_name
                       password
                       ru-ado-const-adconnectunspecified
                 );_ end of list
               );_ end of vl-catch-all-apply
        );_ end of setq
     );_ end of vl-catch-all-error-p
    (progn
     ;; Если обнаружены ошибки, обрабатываем их
      (ru-ado-error-messages
        (ru-ado-error-handler temp_object connection_object)
       nil
     );_ end of ru-ado-error-messages
      (vlax-release-object connection_object)
   );_ end of progn
    (setq result connection_object)
 );_ end of if
 (if result
    (progn
      (setq connection_properties
             (vlax-get-property
               result
               "Properties"
            );_ end of vlax-get-property
     );_ end of setq
     ;;Если в свойствах есть  "Jet OLEDB:ODBC Parsing"
      (if (not (vl-catch-all-error-p
                 (setq connection_parsing_property
                        (vl-catch-all-apply
                          'vlax-get-property
                          (list
                            connection_properties
                            "ITEM"
                            "Jet OLEDB:ODBC Parsing"
                          );_ end of list
                        );_ end of vl-catch-all-apply
                 );_ end of setq
               );_ end of vl-catch-all-error-p
         );_ end of not
;|устанавливаем "Jet OLEDB:ODBC Parsing" в "true" для использования в Jet engine
двойных кавычек вокруг идентификаторов|;
        (vlax-put-property
```

```
          connection_parsing_property
          "VALUE"
          :vlax-true
        );_ end of vlax-put-property
      );_ end of if
    (if (= 'VLA-OBJECT (type connection_parsing_property))
         (vlax-release-object connection_parsing_property)
      );_ end of if
       (if (= 'VLA-OBJECT (type connection_properties))
         (vlax-release-object connection_properties)
      );_ end of if
    );_ end of progn
  );_ end of if
   result
);_ end of defun
```

При возникновении в ADO ошибок они могут быть извлечены из специального объекта `ADODB.Error`. Сообщения об ошибках необходимо извлечь и преобразовать в список. Для этого мы должны написать специальную функцию (листинг 19.4).

Листинг 19.4. Обработчик ошибок ADO

```
(defun ru-ado-error-handler (vl_error_object connection_object /
errors_object error_object error_count error_number error_list
error_value result)
;;; Преобразовывает ошибки в список списков точечных пар
;;; в виде  ("name" . "value")
;;; Вначале обрабатываются возможные сообщения Visual LISP
  (setq result (list (list (cons "Ошибки Visual LISP"
               (vl-catch-all-error-message vl_error_object))))
;;; ADO накапливает ошибки в специальном объекте ADODB.Error,
;;; откуда мы их должны извлечь
        error_object  (vlax-create-object "ADODB.Error")
        errors_object (vlax-get-property connection_object "Errors")
        error_count   (vlax-get-property errors_object "Count")
        error_number  -1
 );_ end of setq
;;; Теперь обрабатываем ошибки errors
  (while (< (setq error_number (1+ error_number)) error_count)
    (setq error_object
               (vlax-get-property errors_object "Item" error_number)
          error_list nil
   );_ end of setq
;;; Проверяем все возможные пунктики ошибок
    (foreach error_property '("Description"
                              "HelpContext"
                              "HelpFile"
                              "NativeError"
                              "Number"
                              "SQLState"
                              "Source"
```

```
                              )
      (if (numberp (setq error_value
                         (vlax-get-property error_object error_property)
                   );_ end of setq
          );_ end of numberp
        (setq error_value (itoa error_value))
     );_ end of if
     (setq error_list (cons (cons error_property error_value)
                            error_list))
   );_ end of foreach
    (setq result (cons (reverse error_List) result))
 );_ end of while
  (reverse result)
);_ end of defun
```

Сообщения об ошибке необходимо показать пользователю. Для этого мы должны написать специальную функцию (листинг 19.5).

Листинг 19.5. Функция `ru-ado-error-messages`

```
(defun ru-ado-error-messages (ado_error_list ado_last_sql /
                              list_names list_values)
;| Вывод сообщения об ошибках, сгенерированного функцией ru-ado-error-handler|;
  (if ado_last_sql
    (setq list_names  (cons "Последний SQL-запрос" list_names)
          list_values (cons ado_last_sql list_values)
   );_ end of setq
 );_ end of if
  (foreach error_list ado_error_list
    (foreach error_item error_list
      (setq list_names  (cons (car error_item) list_names)
            list_values (cons (cdr error_item) list_values)
     );_ end of setq
   );_ end of foreach
 );_ end of foreach
  (ru-dlg-double-list "Информация о сбойной ситуации ADO" "Параметр"
    "Значение" (reverse list_names) (reverse list_values) t)
);_ end of defun
```

Мы воспользовались разработанной нами в *главе 15* функцией вывода двойного списка (рис. 19.5).

После разработки подготовительных функций мы можем разработать и основную функцию, выполняющую SQL-запрос. Но сначала познакомимся с языком SQL[1].

Язык структурированных запросов *SQL* (Structured Query Language) обеспечивает управление структурой БД и манипулирование данными, а также является стандартным средством доступа к удаленным СУБД. В мире СУБД SQL играет примерно такую же роль, как и командная строка в AutoCAD — обеспечивает полный доступ к

[1] Шутка. Для знакомства с SQL пишут книги объемом в несколько сотен страниц.

системе. SQL-запрос, так же, как и выражение, вводимое в командную строку AutoCAD, представляет одну или несколько строк, содержащих специальные выражения. Судя по боевикам, лучше всего языком SQL владеют американские полицейские, набирающие на бортовом компьютере хитроумные запросы, в реальной жизни обычные пользователи пользуются привычными визуальными средствами. Разработчики СУБД просто обязаны владеть языком SQL точно так же, как разработчики приложений для системы AutoCAD — языком LISP.

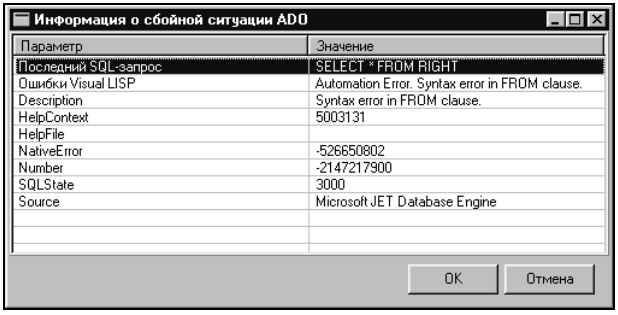


Рис. 19.5. Диалоговое окно информации о сбойной ситуации в ADO

На этом теоретический курс SQL можно считать законченным. Остальное изучим на практике. Сначала разработаем функцию, выполняющую любой SQL-запрос к базе данных через ADO (листинг 19.6). Аргументами функции являются объект `ADODB.Connection` и *строка* SQL-запроса.

Листинг 19.6. Функция `ru-ado-exec-sql`

```
(defun ru-ado-exec-sql (connection_object sql / command_object
 fields_object field_count field_list field_number is_error item
records_affected record_set_list record_set_object result temp_object)
;;; Создаем командный объект
  (setq command_object (vlax-create-object "ADODB.Command"))
;;; Передаем в свойство командного объекта текст SQL-запроса и
;;; соединение с БД
  (vlax-put-property command_object "CommandText" sql)
  (vlax-put-property command_object "ActiveConnection" connection_object)
  (setq record_set_object (vlax-create-object "ADODB.RecordSet"))
;;; Открываем набор данных и пытаемся выловить ошибки
  (if (vl-catch-all-error-p
        (setq temp_object
              (vl-catch-all-apply
                'vlax-invoke-method
                (list command_object
                      "Execute"
                      'records_affected
                      nil
                      ru-ado-const-adcmdtext
               );_ end of list
```

```
              );_ end of vl-catch-all-apply
       );_ end of setq
     );_ end of vl-catch-all-error-p
    (progn
;;; В случае сбойной ситуации выводим сообщение об ошибках
      (ru-ado-error-messages
        (ru-ado-error-handler temp_object connection_object)
        sql
     );_ end of ru-ado-error-messages
      (setq is_error t)
      (vlax-release-object command_object)
      (vlax-release-object record_set_object)
   );_ end of progn
    (setq record_set_object temp_object)
 );_ end of if
;;; Если ошибок нет, обрабатываем данные
  (if (not is_error)
   ;; Если набор закрыт
    (if (= ru-ado-const-adstateclosed
           (vlax-get-property record_set_object "State")
       );_ end of =
     ;; Закрытый набор остается при SQL-запросах,
     ;; содержащих  DELETE, INSERT или UPDATE
     ;; Записи при этом не возвращаются и можно
     ;; удалить объекты
      (progn
        (setq result records_affected)
        (vlax-release-object record_set_object)
        (vlax-release-object command_object)
     );_ end of progn
      (progn
;| Для запросов SELECT вытаскиваем данные полей.
Получаем свойство Fields и количество полей.
Теперь значения полей нужно преобразовать в список. В Delphi мы легко
получали значение любого поля, в LISP это сложнее.
|;
        (setq fields_object
                (vlax-get-property record_set_object "Fields")
              field_count
                           (vlax-get-property fields_object "Count")
              field_number  -1
       );_ end of setq
;;; Составляем список отобранных полей
        (while (> field_count (setq field_number (1+ field_number)))
;;; Составляем список всех полей
          (setq
            field_list
             (cons
               (vlax-get-property
                 (vlax-get-property fields_object "Item" field_number)
                 "Name"
```

```
              );_ end of vlax-get-property
               field_list
             );_ end of cons
          );_ end of setq
        );_ end of while
         (setq result (list (reverse field_list)))
        ;; составили список имен полей
        ;; Если записей несколько
         (if (not (and (= :vlax-true
                          (vlax-get-property record_set_object "BOF")
                       );_ end of =
                        (= :vlax-true
                          (vlax-get-property record_set_object "EOF")
                       );_ end of =
                  );_ end of and
            );_ end of not
           (setq
             result
;;; Сначала добавляем в результат список имен полей
           (append (list (reverse field_list))
;;; Затем добавляем список значений
;|
Здесь мы используем функцию транспонирования списка
по алгоритму Дугласа Вильсона (Douglas Wilson,
 http://xarch.tu-graz.ac.at/autocad/lisp/)
для создания списка записей, т. к.

GetRows возвращает набор данных в виде
(
(ПОЛЕ1_ЗАПИСЬ1 ПОЛЕ1_ЗАПИСЬ2...ПОЛЕ1_ЗАПИСЬX)
(ПОЛЕ2_ЗАПИСЬ1 ПОЛЕ2_ЗАПИСЬ2...ПОЛЕ2_ЗАПИСЬX)
(ПОЛЕX_ЗАПИСЬ1 ПОЛЕX_ЗАПИСЬ2...ПОЛЕX_ЗАПИСЬX)
)
А нам надо в виде
(
(ПОЛЕ1_ЗАПИСЬ1 ПОЛЕ2_ЗАПИСЬ1... ПОЛЕX_ЗАПИСЬ1)
(ПОЛЕ1_ЗАПИСЬ2 ПОЛЕ2_ЗАПИСЬ2... ПОЛЕX_ЗАПИСЬ2)
(ПОЛЕ1_ЗАПИСЬX ПОЛЕ2_ЗАПИСЬX... ПОЛЕX_ЗАПИСЬX)
)
|;
                     (ru-list-Douglas-Wilson-transpose
                       (mapcar
                         '(lambda (record_set_list)
                            (mapcar '(lambda (item)
                                       (ru-ado-variant-to-value item)
                                     );_ end of lambda
                                     record_set_list
                             );_ end of mapcar
                           );_ end of lambda
                           (vlax-safearray->list
                             (vlax-variant-value
                               (vlax-invoke-method
```

```
                                record_set_object
                                "GetRows"
                                ru-ado-const-adgetrowsrest
                              );_ end of vlax-invoke-method
                            );_ end of vlax-variant-value
                          );_ end of vlax-safearray->list
                        );_ end of mapcar
                      );_ end of ru-list-Douglas-Wilson-transpose
              );_ end of append
           );_ end of setq
        );_ end of if
         (vlax-invoke-method record_set_object "Close")
         (vlax-release-object record_set_object)
         (vlax-release-object command_object)
      );_ end of progn
    );_ end of if
  );_ end of if
   result
);_ end of defun
```

Вот и все! Пока, может быть, не очень понятно, но после выполнения нескольких примеров все окажется простым. Обернем эту функцию в более удобную оболочку (листинг 19.7), т. к. удобнее работать со строкой соединения, чем с неведомым объектом.

Листинг 19.7. Функция `ru-ado-exec-sql-to-connection`

```
(defun ru-ado-exec-sql-to-connection (connection_string user_name
                        password SQL / connection_object Result)
;;; Выполнение запросов к любой БД, заданной строкой соединения
(if (setq connection_object (ru-ado-connect-to-db connection_string
               user_name password))
    (progn
      (setq result (ru-ado-exec-sql connection_object SQL))
      (ru-ado-disconnect-from-db connection_object)
    );_ end of progn
  );_ end of if
   result
);_ end of defun
```

Сразу испытаем полученную функцию. Пробуем подключиться к базе данных в формате DBF неизвестно какой версии. Точно знаем, что это один файл ТЕПЛОПУНКТЫ.DBF, лежащий в папке C:\DB\TS.

Создадим глобальную[1] переменную, в которой сохраним строку соединения (разбита на подстроки условно, для улучшения восприятия):

```
(setq my_conn_str (strcat
"Provider=MSDASQL.1;"
"Persist Security Info=False;"
```

[1] Только для тестирования из командной строки!

```
"Extended Properties=DSN=Файлы dBASE;"
"DBQ=C:\\DB\\TS;DefaultDir=C:\\DB\\TS;DriverId=533;"
MaxBufferSize=2048;PageTimeout=5;"))
```

Создадим SQL-запрос:

```
(setq SQL "SELECT * FROM ТЕПЛОПУНКТЫ")
```

Выполняем функцию:

```
(ru-ado-exec-sql-to-connection my_conn_str  "" "" SQL)
```

Получаем результат в виде списка:

```
(
("TP_NUM" "GDE_RASPOL" "TELEFON" "KAM_NUM" "Q_OTOPL" "Q_VENT"
 "Q_GVS_MAX" "Q_GVS_SR" "K_VOSTOK_1" "K_SEVER_1" "ID_ZCAD")
 (1.0 "ЗДС" nil "39/6" 0.0 0.0 0.0 0.0 230.848 -969.221
 "00072716213012")
....
)
```

Первый подсписок — имена полей, далее идут подсписки данных. Пока нас интересовали только имена полей. Выполним еще несколько запросов. Запрашиваем записи по условию с ограничением выводимых полей (вольный перевод: "Выбери номера теплопунктов, расположение, отопительную нагрузку и координаты, при отопительной нагрузке более 5, с сортировкой по отопительной нагрузке. Да поживее!"):

```
(setq SQL "SELECT TP_NUM, GDE_RASPOL, Q_OTOPL, K_VOSTOK_1, K_SEVER_1 FROM
ТЕПЛОПУНКТЫ WHERE [Q_OTOPL] > 5 ORDER BY Q_OTOPL")
(ru-ado-exec-sql-to-connection my_conn_str "" "" SQL)
```

Результат:

```
(("TP_NUM" "GDE_RASPOL" "Q_OTOPL" "K_VOSTOK_1" "K_SEVER_1")
  (14.0 "Красина, 27" 5.172 -345.22 -941.62)
  (150.0 "Мяготина, 42а" 5.219 -2352.58 -843.539)
  (132.0 "5мкр." 5.328 -4586.83 2695.3)
  (111.0 "3мкр., 22" 5.653 -4837.87 2298.39)
  (8.0 "Пушкина, 189а" 5.709 178.035 423.645)
  (74.0 "Мяготина, 61" 6.134 -2356.19 -575.792)
  (65.0 "Конституции, 38" 6.134 0.0 0.0)
  (103.0 "3мкр., 9" 6.422 -5212.13 2637.31)
  (87.0 "7-я больничная" 6.744 -1075.21 -1485.27)
  (112.0 "3мкр., 9" 6.776 -5088.35 1910.71)
  (43.0 "Свердлова, 24а" 10.5 -2186.69 -1201.61)
  (69.0 "Маркса, 4" 15.17 -556.992 -842.729)
  (136.0 "пос. Черемухово" 53.719 -6692.55 -4358.43)
)
```

Это уже ценная информация, особенно координаты. Очевидно, каждый теплопункт мы теперь можем обозначить условным знаком, например, в виде блока с точкой вставки в известных координатах, а масштаб вставки принять пропорционально нагрузке. Совместив это с планом города, мы можем получить почти настоящую тематическую карту, а это уже первый шаг на пути совмещения технологий САПР и ГИС[1].

[1] О геоинформационных технологиях *см. главу 34.*

Продолжим эксперименты. Внесем в базу данных новый вымышленный теплопункт.

```
(setq SQL "INSERT INTO ТЕПЛОПУНКТЫ (TP_NUM, GDE_RASPOL, TELEFON, KAM_NUM, Q_OTOPL,
Q_VENT, Q_GVS_MAX, Q_GVS_SR, K_VOSTOK_1, K_SEVER_1, ID_ZCAD) values (999, 'Центр',
'42-28-36', 'ТК-39А', 120.5, 4.2, 50.0, 30.5, 0.0, 0.0, 'RU_CAD_DEMO')")
(ru-ado-exec-sql-to-connection my_conn_str  "" "" SQL)
```

Проверим, добавлена ли новая запись:

```
(setq SQL "SELECT TP_NUM, GDE_RASPOL, Q_OTOPL, K_VOSTOK_1, K_SEVER_1 FROM
ТЕПЛОПУНКТЫ WHERE TP_NUM=999.0")
(ru-ado-exec-sql-to-connection my_conn_str  "" "" SQL)
```

Получаем данные по добавленному объекту:

```
(("TP_NUM" "GDE_RASPOL" "Q_OTOPL" "K_VOSTOK_1" "K_SEVER_1")
  (999.0 "Центр" 120.5 0.0 0.0)
)
```

Изменим координаты объекта:

```
(setq SQL "UPDATE ТЕПЛОПУНКТЫ SET K_VOSTOK_1=100.0 WHERE TP_NUM=999.0")
(ru-ado-exec-sql-to-connection my_conn_str "" "" SQL)
(setq SQL "SELECT TP_NUM, GDE_RASPOL, Q_OTOPL, K_VOSTOK_1, K_SEVER_1 FROM
ТЕПЛОПУНКТЫ WHERE TP_NUM=999.0")
(ru-ado-exec-sql-to-connection my_conn_str "" "" SQL)
```

Получаем измененные данные по добавленному объекту:

```
(("TP_NUM" "GDE_RASPOL" "Q_OTOPL" "K_VOSTOK_1" "K_SEVER_1")
  (999.0 "Центр" 120.5 100.0 0.0)
)
```

Удаляем вымышленный теплопункт:

```
(setq SQL "DELETE  FROM ТЕПЛОПУНКТЫ WHERE TP_NUM=999.0")
(ru-ado-exec-sql-to-connection my_conn_str "" "" SQL)
```

Разработка конструктора строки соединения

Мы уже размышляли о необходимости собственного конструктора строки соединения с любой БД. Такое приложение сделать проще, чем разыскать в закоулках Windows подходящую утилиту. Не останавливаясь на многократно изученных деталях, скажем только, что это приложение имеет вид, показанный на рис. 19.6, важные фрагменты кода приведены в листинге 19.8, а функция для вызова конструктора из LISP-функций — в листинге 19.9.

Листинг 19.8. Фрагменты приложения-конструктора строки

```
unit frmConnConstructor;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
 Dialogs, ExtCtrls, StdCtrls, ComCtrls, Mask, JvToolEdit, ActnList;
```

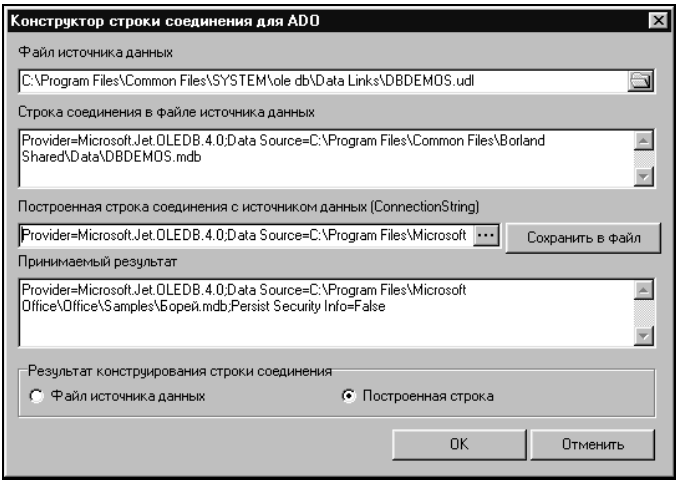


Рис. 19.6. Конструктор строки соединения

```
type
  TfrmConstructor = class(TForm)
    MemoFromUDL: TMemo;
    editUdlFileName: TJvComboEdit;
    editConnectionString: TJvComboEdit;
    btnSaveToUdl: TButton;
    RadioGroupResult: TRadioGroup;
    btnOk: TButton;
    btnCancel: TButton;
    ActionList: TActionList;
    aSelectUdlFile: TAction;
    aBuildConnString: TAction;
    aSaveToUdl: TAction;
    aOk: TAction;
    aCancel: TAction;
    MemoResult: TMemo;
    SaveDialog: TSaveDialog;
    procedure editUdlFileNameButtonClick(Sender: TObject);
    procedure ActionListUpdate(Action: TBasicAction; var Handled: Boolean);
    procedure aSelectUdlFileExecute(Sender: TObject);
    procedure aBuildConnStringExecute(Sender: TObject);
    procedure aSaveToUdlExecute(Sender: TObject);
    procedure aOkExecute(Sender: TObject);
    procedure aCancelExecute(Sender: TObject);
    procedure RadioGroupResultClick(Sender: TObject);
    procedure editConnectionStringButtonClick(Sender: TObject);
    procedure FormShow(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
  public
    ConnectionString: WideString;
  end;
```

```
var
  frmConstructor: TfrmConstructor;

implementation

uses ADODB, JclUnicode, ruUtils, OLEDB, ComObj;

{$R *.DFM}
function GetChar(Value: String; index: smallint): Char;
begin
   if Length(Value) < index then result := #0 else Result := Value[index];
end;

function AsChar(Value: String): Char;
begin
  result := GetChar(Value,1);
end;

Function ReadUDL(Filename: String): String;
var
   UDLList: TWideStringList;
   ConStr: String;
   i: Integer;
   FirstChar: Char;
begin
{
Чтение UDL-файла для отображения строки соединения в Memo.

}
   result := '';
   If FileExists(FileName) then begin
      UDLList := TWideStringList.Create;
      Try
         UDLList.LoadFromFile(Filename);
         For i := 0 to UDLList.Count - 1 do begin
            FirstChar := AsChar(TrimLeft(UDLList[i]));
            If not (FirstChar in ['[',';']) then
               ConStr := ConStr + Trim(UDLList[i]);
         end;
      finally
         UDLList.Free;
         result := ConStr;
      end;
   end;
end;

procedure TfrmConstructor.editUdlFileNameButtonClick(Sender: TObject);
begin
aSelectUdlFileExecute(Sender);
end;

procedure TfrmConstructor.aSelectUdlFileExecute(Sender: TObject);
var
UdlFileName:string;
Begin
```

```
{
Вызов стандартного диалогового окна выбора UDL-файла (рис. 19.7)
}
  UdlFileName:= PromptDataLinkFile(Handle, editUdlFileName.Text);
  editUdlFileName.Text:= UdlFileName;
// Просто так UDL не читается!!
//  MemoFromUDL.Lines.LoadFromFile(UdlFileName);
  MemoFromUDL.Text := ReadUDL(UdlFileName);
  RadioGroupResultClick(Sender);
end;

procedure TfrmConstructor.aBuildConnStringExecute(Sender: TObject);
begin
   ConnectionString := editConnectionString.Text;
{
Вызов стандартного диалогового окна построения соединения (рис. 19.5-19.8)
}
   ConnectionString := PromptDataSource(Handle, ConnectionString);
   If ConnectionString <> '' then
      editConnectionString.Text := ConnectionString;
  RadioGroupResultClick(Sender);
end;

procedure TfrmConstructor.aSaveToUdlExecute(Sender: TObject);
var
  UdlFileName: WideString;
  DataInit: IDataInitialize;
begin
   If SaveDialog.Execute then begin
{
 Вот как надо записывать UDL-файл, не пытаясь записать в него данные,
как в INI-файл, или в виде обычного текста.
}
      DataInit := CreateComObject(CLSID_DataLinks) as IDataInitialize;
      UdlFileName := ChangeFileExt(SaveDialog.FileName,'.udl');
      if FileExists(UdlFileName) then DeleteFile(UdlFileName);
     OleCheck(DataInit.WriteStringToStorage(PWideChar(UdlFileName),
     PWideChar(ConnectionString), CREATE_NEW));
   end;
end;

procedure TfrmConstructor.RadioGroupResultClick(Sender: TObject);
var
 CS : WideString;
begin
   MemoResult.Text :='';
   if   RadioGroupResult.ItemIndex = 0 then
      begin
      if editUdlFileName.Text <>'' then
      CS := 'FILE NAME='+editUdlFileName.Text;
      end else
```

```
      CS := editConnectionString.Text;
    MemoResult.Text := CS;
    {if CS <> '' then} ConnectionString:=CS;
end;

procedure TfrmConstructor.editConnectionStringButtonClick(Sender: TObject);
begin
aBuildConnStringExecute(Sender);
end;
end.
```

Листинг 19.9. Функция `ru-ado-build-connection-string`

```
(defun ru-ado-build-connection-string ( / srv result)
;;; Построение ConnectionString
(if (setq srv (vlax-get-or-create-object
                        "ruADOConnSvr.ConnStrConstructor"))
    (setq result (vlax-invoke-method srv "Execute"))
    (princ "\nНе удалось запустить ruADOConnSvr.ConnStrConstructor")
 );_ end of if
  result
);_ end of defun
```

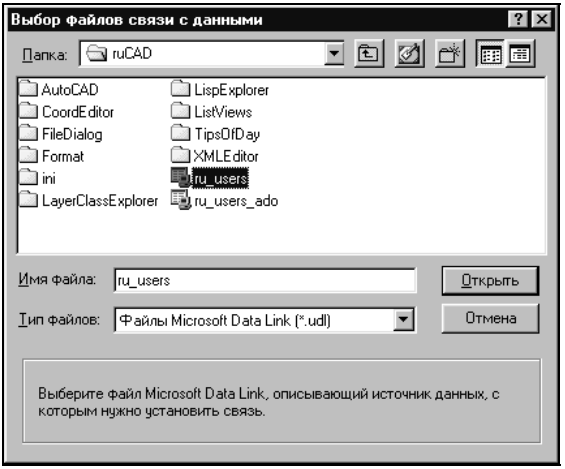


Рис. 19.7. Системное диалоговое окно выбора UDL-файла

Полный пример работы с БД

Теперь сделаем демонстрационную программу, которая практической ценности не имеет, но может послужить прототипом для разработки реальных приложений. Но предварительно необходимо решить вопрос с получением списка имеющихся в БД таблиц. Если мы работаем со своей БД, то знаем имена и структуру таблиц. Для чужой БД потребуется извлечение списка таблиц. Кроме физически существующих

таблиц в базе данных могут быть и виртуальные таблицы, называемые *просмотрами* (Views). Просмотр (View) — это виртуальная таблица, созданная на основе запроса к обычным таблицам. Просмотр реализован как SQL-запрос, хранящийся в базе данных и выполняющийся каждый раз, когда происходит обращение к представлению. Просмотр может выбирать данные из нескольких таблиц. Для получения списка таблиц и просмотров разработаем небольшую функцию (листинг 19.10).

Листинг 19.10. Функция `ru-ado-get-tables-and-views`

```
(defun ru-ado-get-tables-and-views (connection_object / temp_object
tables_list tmp_list views_list)
  (setq record_set_object (vlax-create-object "ADODB.RecordSet"))
  (if (vl-catch-all-error-p
        (setq record_set_object
               (vl-catch-all-apply
                 'vlax-invoke-method
                 (list
                   connection_object
                   "OpenSchema"
                   ru-ado-const-adschematables
                 );_ end of list
               );_ end of vl-catch-all-apply
      );_ end of setq
    );_ end of vl-catch-all-error-p
    (ru-ado-error-messages
      (ru-ado-error-handler temp_object connection_object)
      sql
   );_ end of ru-ado-error-messages
    (progn
      (setq
        tmp_list
        (ru-list-Douglas-Wilson-transpose
          (mapcar
            '(lambda (InputList)
               (mapcar '(lambda (Item)
                          (ru-ado-variant-to-value Item)
                        );_ end of lambda
                        InputList
               );_ end of mapcar
            );_ end of lambda
            (vlax-safearray->list
              (vlax-variant-value
                (vlax-invoke-method
                  record_set_object
                  "GetRows"
                  ru-ado-const-adgetrowsrest
                );_ end of vlax-invoke-method
              );_ end of vlax-variant-value
            );_ end of vlax-safearray->list
          );_ end of mapcar
```

```
        );_ end of ru-list-Douglas-Wilson-transpose
     );_ end of setq
      (foreach Item tmp_list
        (cond
          ((= (nth 3 Item) "VIEW")
           (setq views_list (cons (nth 2 Item) views_list))
          )
          ((= (nth 3 Item) "TABLE")
           (setq tables_list (cons (nth 2 Item) tables_list))
          )
        );_ end of cond
     );_ end of foreach
      (vlax-invoke-method record_set_object "Close")
   );_ end of progn
 );_ end of if
  (vlax-release-object record_set_object)
  (list tables_list views_list)
);_ end of defun
```

Исходный текст демонстрационной программы приведен в листинге 19.11.

Листинг 19.11. Демонстрационная программа для работы с любой БД

```
(defun c:ru_ado_demo (/ connection_object connection_string
 list_tables_and_views list_tables list_views table table_list view
 view_list)
;;; Пример использования ADO
  (ru-msg-info "Шаг 1. Строим ConnectionString")
  (print (setq connection_string (ru-ado-build-connection-string)))
;|
Давайте соединимся с демонстрационной базой данных Борей.mdb, входящей в комплект
Microsoft Office. Эта база подходит для тестирования потому, что она достаточно
сложна, в ней имеются не только плоские таблицы, но и связи между ними.
Строка соединения будет иметь вид:
Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\Program Files\Microsoft
Office\Office\Samples\Борей.mdb;Persist Security Info=False
|;
  (if (/= connection_string "")
    (progn
      (ru-msg-info
        (strcat "Шаг 2. Соединяемся с БД, используя ConnectionString:\n"
                connection_string))
      (if (setq connection_object (ru-ado-connect-to-db connection_string
                  "admin" ""))
        (progn
          (ru-msg-info "Шаг 3. Извлекаем список таблиц и просмотров")
          (print (setq list_tables_and_views
                       (ru-ado-get-tables-and-views
                         connection_object
                       );_ end of ru-ado-get-tables-and-views
                 );_ end of setq
          );_ end of print
```

```
;|
Получили список таблиц и просмотров в БД Борей
(
;;;Список таблиц:
 ("Товары" "Типы" "Сотрудники" "Поставщики" "Клиенты" "Заказы" "Заказано"
"Доставка"
)
;;;Список просмотров:
  ("Товары с ценой выше средней" "Счета" "Список имеющихся товаров" "Сведения
о заказах" "Промежуточная сумма заказа" "Продажи товаров в 1995" "Продажи по типам
в 1995" "Продажи по типам" "Квартальные обороты"
  "Десять самых дорогих товаров"
 )
)
|;
;; Выбор таблицы (рис. 19.8)
          (if (setq list_tables (car list_tables_and_views))
            (progn
              (print (setq table (ru-dlg-single-list
                                   "Шаг 4. Выбираем таблицу БД"
                                   "Таблицы в БД"
                                   list_tables
                                 );_ end of ru-dlg-single-list
                     );_ end of setq
              );_ end of print
;;; Выбрали таблицу "Сотрудники"
              (if table
                (print (setq table_list
                             (ru-ado-exec-sql
                               connection_object
                               (strcat "SELECT * FROM " table)
                             );_ end of ru-ado-exec-sql
                       );_ end of setq
                );_ end of print
              );_ end of if
            );_ end of progn
            (ru-msg-alert "Таблицы не найдены")
          );_ end of if
```

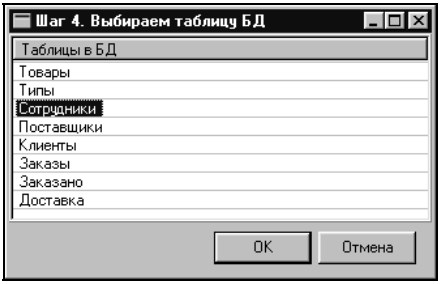


Рис. 19.8. Выбор таблицы

```
;|
Получили таблицу в виде списка (приводим фрагмент)
(
;; Сначала подсписок имен полей
("КодСотрудника" "Фамилия" "Имя" "Должность" "Обращение"
   "ДатаРождения" "ДатаНайма" "Адрес" "Город" "Область" "Индекс"
   "Страна" "ДомашнийТелефон" "Добавочный" "Фотография" "Примечания"
   "Подчиняется")
Далее список данных по каждому сотруднику:
  (1 "Белова" "Мария" "Представитель" "г-жа" 25180.0 33725.0
    "ул. Нефтяников, 14-4" "Москва" nil "122981" "Россия"
    "(095) 555-9857" "124-5467" #<safearray...>
    "Окончила Институт Пищевой промышленности в 1991 г. Работала продавцом в
киоске, так что имеет большие навыки в торговле продовольственными товарами.
Отличается исключительным добродушием и мягким характером." 2
 )
  ...
  (9 "Ясенева" "Инна" "Представитель" "г-жа" 24134.0 34653.0
    "Родниковый пер. 1" "Киев" nil "255321" "Украина" "нет" "314-0452"
    #<safearray...>
    "Очень хорошая сотрудница - старательная, добрая, отзывчивая, технически
грамотная и во всех отношениях аккуратная. Еще не было ни одного начальника,
который был бы хоть чем-то недоволен..." 2
 )
)
Можно заметить, что даты представлены в неудобном формате. Устранить этот
недостаток легко. Достаточно переопределить функцию (ru-ado-variant-to-value item),
вызываемую в ru-ado-exec-sql. Предоставляем читателям, в порядке домашнего задания,
сделать это самостоятельно.
Фотографии сотрудников, хранящиеся в БД в BLOB-полях, показаны в виде ссылки
#<safearray...>. При желании можно извлечь и фотографии, но это уже не тема книги
по САПР.

|;
          (if (setq list_views (cadr list_tables_and_views))
            (progn
;; Выбор просмотра (рис. 19.9)
```

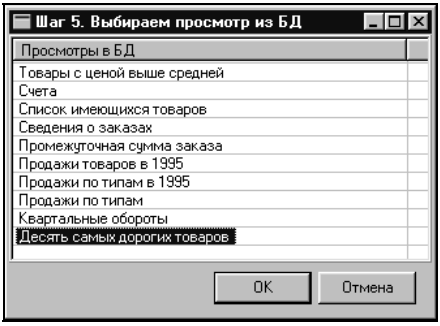


Рис. 19.9. Выбор просмотра

```
              (print (setq view (ru-dlg-single-list
                                  "Шаг 5. Выбираем просмотр из БД"
                                  "Просмотры в БД"
                                  list_views
                                );_ end of ru-dlg-single-list
                     );_ end of setq
               );_ end of print
;;;Выберем "Десять самых дорогих товаров"
               (if view
                 (print (setq view_list
                              (ru-ado-exec-sql
                                connection_object
                                (strcat "SELECT * FROM [" view "]")
                              );_ end of ru-ado-exec-sql
;|К просмотру, как и к обычной таблице, мы обратились по имени. Имя просмотра
мы заключили в квадратные скобки, т. к. оно содержит пробелы.
|;
                       );_ end of setq
                );_ end of print
               );_ end of if
             );_ end of progn
              (ru-msg-alert "Просмотры не найдены")
           );_ end of if
;| Получен список самых дорогих товаров. Да, фирма "Борей" явно не веники вяжет!
(("СамыеДорогиеТовары" "Цена")
  ("Cote de Blaye" 1.18575e+006)
  ("Thuringer Rostbratwurst" 557055.0)
  ("Mishi Kobe Niku" 436500.0)
  ("Sir Rodney's Marmalade" 364500.0)
  ("Carnarvon Tigers" 281250.0)
  ("Raclette Courdavault" 247500.0)
  ("Manjimup Dried Apples" 238500.0)
  ("Tarte au sucre" 221850.0)
  ("Ipoh Coffee" 207000.0)
  ("Rossle Sauerkraut" 205200.0)
)
|;
             (ru-msg-info "Шаг 6. Отключаемся от БД")
             (ru-ado-disconnect-from-db connection_object)
           );_ end of progn
            (ru-msg-alert "Не создан объект соединения")
         );_ end of if
       );_ end of progn
        (ru-msg-alert "Не построена строка соединения")
     );_ end of if
      (princ)
);_ end of defun
```

Итак, мы достаточно подробно исследовали механизмы работы с базами данных. Надеемся, что, используя приведенные схемные решения, читатели смогут самостоятельно разработать собственные функции.

Применимость технологии работы с базами данных

А теперь подумаем, нужно ли все это в нашей системе. На этапе постановки задачи мы предусматривали ведение списка пользователей системы в базе данных формата MS Access *(см. главу 3)*. Эта задумка была реализована и даже подробно описана в первом варианте данной главы. В дальнейшем нам пришлось от этой идеи отказаться из-за надуманности и ненадежности решения, а учет пользователей мы перенесли в реестр Windows (о причинах мы расскажем в *главе 36*).

Мы могли бы хранить множество технических данных, необходимых для рисования и расчетов (например, сортаменты проката и труб) в таблицах БД. Но эти данные нам нужны для конкретных целей — обычно для рисования. Для того чтобы нарисовать прокатный профиль, мы должны выбрать из сортамента конкретный типоразмер, и выбор должен быть непременно визуальный. Сделать в LISP это можно, но очень неудобно. Сначала, с помощью описанной выше технологии нужно выбрать данные из таблицы и поместить в список, потом как-то отобразить этот список и сделать выбор. Для разнообразных данных это потребует разработки множества индивидуальных программ. Лучше уж не усложнять доступ к данным и хранить их прямо в виде списка, как многие и делают.

Но эту задачу мы уже решили с использованием XML-меню, путем разработки единственной DLL, позволяющей и отображать любые данные, и редактировать их, и экспортировать в AutoCAD. Точно так же можно работать и с базами данных — просматривать, редактировать и выбирать с помощью более удобных программ, написанных на Delphi или Visual Basic, а в LISP отправлять только результаты, необходимые для рисования.

В *главе 32* мы еще рассмотрим технологию выпуска спецификаций оборудования, основанную на работе с настоящей клиент-серверной базой данных. Там основная работа происходит в Delphi-приложении, а в системе AutoCAD для отрисовки LISP-программой отправляются только подготовленные данные.

Еще один важный аспект. Мы привели пример добавления, удаления и редактирования данных в одной таблице ТЕПЛОПУНКТЫ, но не рискнули делать это с базой данных Борей. Это настоящая база данных, а не "плоский" файл. В БД Борей мы обнаружили восемь таблиц и десять просмотров, между ними наверняка установлены связи, о которых мы ничего не знаем, поэтому должна производиться проверка целостности данных. Все это предусмотрено в специальном демонстрационном приложении, зашитом внутрь самой базы данных. Залезать внутрь этой сложной системы с грубыми запросами INSERT, UPDATE или DELETE было бы неразумно.

Работать с базами данных из Visual LISP можно и нужно, но только хорошо продумав необходимость решения такой задачи именно в LISP. Редактировать БД Борей из LISP практически невозможно, но такая задача легко решается и в самой среде MS Access, и в программах, написанных на Delphi или Visual Basic. Редактировать таблицу ТЕПЛОПУНКТЫ можно во множестве программ, но как заполнять поля координат объекта? Вычислять графоаналитическими методами? А в системе AutoCAD это сделать просто — получить координаты точки указанием на плане города и отправить в базу данных примерно так, как было показано в примере. Про-

чие, не пространственно координированные данные, удобнее редактировать в специализированном визуальном редакторе.

Резюме

Мы подробно рассмотрели технологию работы с базами данных из Visual LISP, выявили ее возможности, достоинства и недостатки. В разрабатываемой системе эти технологии мы будем применять ограниченно, но наши читатели могут найти им подходящее применение.

Уже завершив работу над этой главой, мы все-таки нашли в Интернете специалистов, занимающихся работой с ADO из LISP. Это *Fleming Consulting Group*[1] — небольшая, кажется, семейная фирма (Jon и Barbara Fleming), разработавшая библиотеку *ADOLISP Library* для работы с ADO из Visual LISP. Изложенные в этой главе решения алгоритмически почти совпадают с ADOLISP Library, и это не удивительно, т. к. технология ADO является универсальной.

Очень жаль, что разыскали мы *ADOLISP Library* слишком поздно, практически мы "изобрели велосипед", но некоторые участки кода этой библиотеки мы использовали в своих функциях. Заодно мы убедились и в правильности принятых нами решений.

[1] **www.fleming-group.com**.

ГЛАВА 20

Разработка программы-стартера

До сих пор мы запускали систему щелчком по ярлыку. При этом запускалась система AutoCAD с заданным профилем *ruCAD*. На первом месте в списке каталогов поддержки для нашего профиля находился наш каталог, в котором лежит файл acaddoc.lsp. Автоматическая загрузка этого файла приводила к автоматической загрузке наших библиотек и инициализации системы. Напомним, что в файле acaddoc.lsp было записано присваивание `(setq *ru_root_dir* "c:\\.ru\\cad\\")`, без которого дальнейшая работа была бы невозможна. Все это работает, но годится только как временный вариант.

В *главе 3* мы наметили разработку программы-стартера, которая должна:

- осуществлять идентификацию пользователя, установку его предпочтений, привилегий и разрешений;
- выполнять основные настройки системы и реестра пользователей;
- выбирать рабочую версию системы AutoCAD;
- запускать выбранную версию системы AutoCAD (возможно, с предварительным выбором документов для загрузки);
- генерировать файл для автоматической загрузки требуемых библиотек;
- выступать в качестве внешнего COM-сервера, к объектам и методам которого можно обращаться из LISP-программ.

Кроме того, стартер может выступать своеобразным Центром управления, из которого могут запускаться любые программы, причем надо сделать так, чтобы подключение любых программ можно было делать без переделки самого стартера. Этот "коварный план" позволит, при желании, единственным ярлыком на рабочем столе сделать ярлык нашего стартера-интегратора.

С учетом наших предварительных разработок сделать все это на удивление просто. Внешний вид стартера показан на рис. 20.1.

Работа в стартере

При запуске стартера выводится хорошо знакомое нам диалоговое окно регистрации пользователя. В случае успешной регистрации пользователь заходит в систему. Если

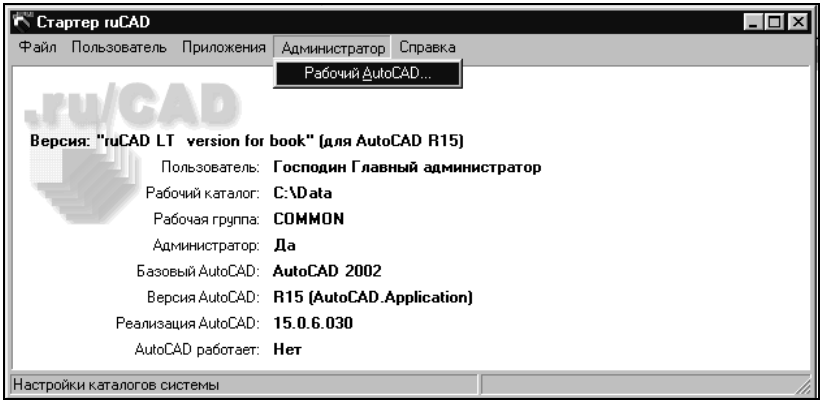


Рис. 20.1. Стартер системы

пользователь обладает правами администратора, ему доступны все пункты меню. Находясь в стартере, пользователь может перерегистрироваться под другим именем и изменить личные настройки. Администратор может изменить настройки любого пользователя, добавлять и удалять пользователей.

Администратор может выбрать рабочую версию AutoCAD, настроить меню дополнительных приложений, а обычный пользователь — запускать приложения. Ну, и прочие мелочи, наподобие получения ненужных советов, справок.

Основное назначение стартера — запуск системы AutoCAD, возможно, с открытием заданного файла. При запуске посредством ярлыка, к которому не привязано имя файла рисунка, система AutoCAD создает новый пустой рисунок, что весьма раздражает некоторых пользователей. Мы не зря создавали комфортабельное диалоговое окно выбора файлов, самым подходящим местом для его использования как раз и является стартер.

Меню **Файл** стартера мы дополним списком последних открытых файлов (рис. 20.2).

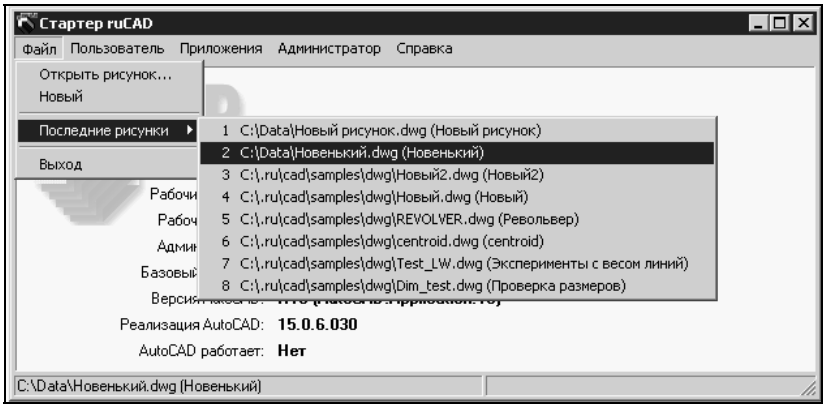


Рис. 20.2. Список последних загружавшихся рисунков

Обратите внимание, что в списке указано не только полное имя файла, но и комментарий к файлу. Пользователь имеет возможность с помощью пункта **Открыть рисунок** меню **Файл** выбрать существующий файл или создать в заданном месте новый файл с заданным именем, воспользовавшись пунктом меню **Новый**, создать новый рисунок с именем по умолчанию (часто это применяется для "бросовых" рисунков, которые не нужно сохранять), выбрать рисунок из списка **Последние рисунки**. Если программу-стартер запустить с параметром в виде имени файла, то сразу после регистрации запустится система AutoCAD.

Прежде чем разбирать программы, остановимся только на ключевых моментах.

Выбор рабочей версии AutoCAD

Обычно на компьютере у конечного пользователя установлена одна версия AutoCAD, но у "особо одаренных" в наличии может быть несколько комплектов. Мы разрабатываем свою систему, как минимум, для двух базовых версий: AutoCAD 2002 (R15) и AutoCAD 2004 (R16), и должны учитывать, что любая из них (или все вместе) могут использоваться в работе. При инсталляции системы ruCAD будет выбираться рабочая версия AutoCAD, но администратор должен иметь возможность подключения к любой версии. Выбор рабочей версии производится в диалоговом окне **Выбор AutoCAD** (рис. 20.3).

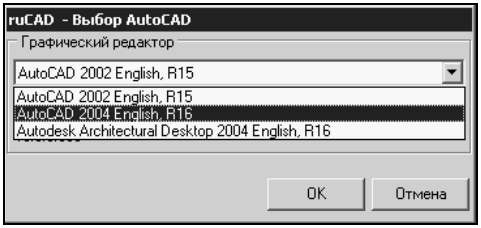


Рис. 20.3. Выбор рабочей версии AutoCAD

Для отображения списка зарегистрированных в операционной системе версий системы AutoCAD использован компонент `ComboBox`. Список установленных версий системы AutoCAD получаем с помощью процедуры `ruGetACADExePathNames` (листинг 20.1), вынесенной в модуль ruAcad.pas.

Листинг 20.1. Получение списка установленных версий системы AutoCAD

```
procedure ruGetACADExePathNames(var lst: TStringList);
{Локальная процедура получения списка для конкретной версии
}
  procedure GetAcadReleaseList(const Rel: string);
  var
    r: TRegistry;
    s, rList: TStringList;
    i, c: integer;
  begin
    r := TRegistry.Create;
```

```
    r.RootKey := HKEY_LOCAL_MACHINE;
    s := TStringList.Create;
    rList := TStringList.Create;
    if r.OpenKey(keyAcadRegRoot, false)
      then begin
      r.GetKeyNames(s);
      r.CloseKey;
      for i := 0 to s.Count - 1 do
        if (AnsiStrLIComp(pchar(Rel), pchar(s[i]), length(Rel)) = 0)
          then rList.Add(keyAcadRegRoot + '\' + s[i]);
      for i := 0 to rList.Count - 1 do begin
        r.CloseKey;
        r.OpenKey(rList[i], false);
        s.Clear;
        r.GetKeyNames(s);
        for c := 0 to s.Count - 1 do begin
          if (AnsiStrLIComp('ACAD', pchar(s[c]), 4) = 0)
            then begin
            r.CloseKey;
            if r.OpenKey(rList[i] + '\' + s[c], false)
              then begin
                lst.Add(copy(rList[i],
                (LastDelimiter('\', rList[i]) + 1), maxInt) + '\' +
                s[c] + '=' +
                r.ReadString('ProductName') + ' ' +
                r.ReadString('Language') + ', ' + Rel);
            end;
          end;
        end;
      end;
    end;
    s.Free;
    rList.Free;
    r.CloseKey;
    r.Free;
  end;
begin
{Получение списка для версий R15 и R16. Появится R18 - достаточно будет добавить
одну строчку}
  GetAcadReleaseList('R15');
  GetAcadReleaseList('R16');
end;
```

Исходный текст модуля frmSelectAcad.pas, в котором реализовано диалоговое окно выбора графических редакторов, приведен в листинге 20.2.

Листинг 20.2. Файл frmSelectAcad.pas

```
unit frmSelectAcad;

interface
```

```
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Buttons, ExtCtrls;

type
  TAcadSelectDlg = class(TForm)
    GroupBox1: TGroupBox;
    ComboBoxAutoCADs: TComboBox;
    bOK: TButton;
    bCanc: TButton;
    Panel2: TPanel;
    labelAcadFile: TLabel;
    LabelProductRelease: TLabel;
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure ComboBoxAutoCADsChange(Sender: TObject);
    procedure bOKClick(Sender: TObject);
    procedure FormShow(Sender: TObject);
  private
  public
{Public-переменные нужны нам для того, чтобы в основном приложении получить
основные параметры выбранной версии системы AutoCAD
}
      AcadVersion, AcadProductRelease, AcadProductName,
      AcadExeName : string;
  end;

function ruSelectAcad: boolean;

implementation

{$R *.DFM}

uses ruUtils, ruStrObj, registry, ruFileUtils, ruConst, ruAcad;

function ruSelectAcad: boolean;
  var
    f: TAcadSelectDlg;
begin
  f:=TAcadSelectDlg.Create(Application);
  Result:=(f.ShowModal=mrOk);
  f.Free;
end;

procedure TAcadSelectDlg.FormCreate(Sender: TObject);
  var
    StringListAcadNames: TStringList;
    i, c: integer;
    AcadKey: string;
begin
  AcadKey:=ruGet_AcadKey;
  labelAcadFile.Caption:='';
```

```
  AcadExeName :='';
  StringListAcadNames:=TStringList.Create;
  ruGetACADExePathNames(StringListAcadNames);
{
 Список строк вида "Такой-то Автокад Язык=R15
}
 ComboBoxAutoCADs.Items.Clear;
  ComboBoxAutoCADs.Enabled:=StringListAcadNames.Count<>0;
  c:=-1;
  for i:=0 to StringListAcadNames.Count-1 do begin
     ComboBoxAutoCADs.Items.AddObject
      (StringListAcadNames.Values[StringListAcadNames.Names[i]],
      TStr.CreateStr(StringListAcadNames.Names[i]));
      if (StrComp(pchar(AcadKey), pchar(StringListAcadNames.Names[i]))=0)
      then c:=i;
    end;
  StringListAcadNames.Free;
  ComboBoxAutoCADs.ItemIndex:=c;
end;

procedure TAcadSelectDlg.FormDestroy(Sender: TObject);
  var
    i: integer;
begin
  for i:=0 to ComboBoxAutoCADs.Items.Count-1 do begin
    TStr(ComboBoxAutoCADs.Items.Objects[i]).Free;
  end;
end;

procedure TAcadSelectDlg.ComboBoxAutoCADsChange(Sender: TObject);
  var
    r: TRegistry;
    n: string;
    RegS : String;
begin
  if (ComboBoxAutoCADs.ItemIndex<0) or
     (ComboBoxAutoCADs.Items.Count=0)
    then begin
      labelAcadFile.Caption:='Автокад не найден';
      AcadExeName :='';
      AcadVersion  :='';
      AcadProductRelease:='';
      AcadProductName:='';
      LabelProductRelease.Caption:=AcadProductRelease;
      Exit;
    end;
  n:='';
  r:=TRegistry.Create;
  r.RootKey:=HKEY_LOCAL_MACHINE;
  if r.OpenKey(keyAcadRegRoot, false)
```

```
    then begin
      RegS :=
 TStr(ComboBoxAutoCADs.Items.Objects[ComboBoxAutoCADs.ItemIndex]).Value;
      AcadVersion  := Copy(RegS,2,2);
      if r.OpenKey(RegS, false)
        then begin
         n:=r.ReadString('AcadLocation');
         AcadProductRelease:=r.ReadString('Release');
         AcadProductName:=r.ReadString('ProductName');
        end;
      if n<>''
        then n:=ruIncludeTrailingBackslash(n)+'acad.exe'
        else n:='';
    end;
  r.CloseKey;
  r.Free;
  if (n<>'') and (FileExists(n))
    then begin
    labelAcadFile.Caption:=n;
      LabelProductRelease.Caption:=AcadProductRelease;
    AcadExeName :=n;
    end
    else begin labelAcadFile.Caption:='Автокад не найден';
     AcadExeName :='';
     end;
end;

procedure TAcadSelectDlg.bOKClick(Sender: TObject);
begin
  if (ComboBoxAutoCADs.Items.Count>0) and
     (ComboBoxAutoCADs.ItemIndex>=0)
  then begin
{ Параметры выбранной системы AutoCAD мы записываем в собственный ключ реестра
для текущего пользователя
}
   if AcadExeName <> '' then
     ruRegWriteParam(keyAcadExeKey,AcadExeName);
     ruRegWriteParam(keyAcadProductName,AcadProductName);
     ruRegWriteParam(keyAcadProductRelease,AcadProductRelease);
     ruRegWriteParam(keyAcadVersion,AcadVersion);
     ruRegWriteParam(keyAcadKey,
          TStr(ComboBoxAutoCADs.Items.Objects
          [ComboBoxAutoCADs.ItemIndex]).Value);
 end;
end;

procedure TAcadSelectDlg.FormShow(Sender: TObject);
begin
  ComboBoxAutoCADsChange(Sender);
end;
end.
```

Запуск AutoCAD

Теперь в основном модуле программы-стартера нам известно о выбранной системе AutoCAD почти все (о том, что мы не знаем самого главного, мы пока еще не знаем). Но, прежде чем писать текст программы, давайте рассмотрим, как мы можем запускать AutoCAD.

Во-первых, мы можем запустить acad.exe как обычный процесс. Это самый простой и надежный способ, тем более что система AutoCAD поддерживает множество переключателей командной строки *(см. главу 3)*. Недостаток этого способа в том, что после запуска AutoCAD основная программа забывает о его существовании. Конечно, мы можем предусмотреть вариант запрета запуска второй копии AutoCAD или, наоборот, разрешить запуск других экземпляров, но при наличии многодокументного режима работы — это излишняя роскошь.

Во-вторых, мы можем импортировать библиотеку типов, получить интерфейсный файл Autocad_TLB.pas, включить его в `uses` и обращаться к объектной модели AutoCAD в полном объеме. Это, на первый взгляд, кажется наилучшим решением. Мы не только сможем запустить систему AutoCAD, но даже и рисовать в ней, нисколько не хуже, чем из приложений, разрабатываемых на встроенном в AutoCAD VBA. Однако объектная модель имеет и ряд недостатков. Например, мы могли бы написать такой код для запуска системы AutoCAD и загрузки в нее заданного файла:

```
Uses AutoCAD_TLB.pas;
...
function ruGetOrCreateAcadObject(AcadAppString: string)
                                                : IAcadApplication;
begin
  try
    Result := GetActiveOleObject(AcadAppString) as IAcadApplication;
  except
    try
      Result := CreateOleObject(AcadAppString) as IAcadApplication;
    except
      ShowMessage ('Не удалось запустить AutoCAD');
    end;
  end;
end;
procedure RunAcadAsIAcadApplication (DwgName:string);
var AcadApplication : IacadApplication;

begin
{Для простоты опускаем обработку ошибок}
 AcadApplication := ruGetOrCreateAcadObject('AutoCAD.Application');
 if FileExists(DwgName) then
   AcadApplication.Documents.Open(DwgName, False)
   else begin
     AcadApplication.Documents.Add('ACADISO');
     AcadApplication.ActiveDocument.SaveAs(DwgName, acNative);
   end;
end;
```

Вроде бы все правильно, но результат будет омерзительным. После выполнения функции `ruGetOrCreateAcadObject` запустится система AutoCAD, которая создаст пустой рисунок, выполнит все, что мы предусмотрели в acaddoc.lsp, включая все настройки и вывод диалогового окна свойств рисунка (а это нам совсем не требуется). Только потом это же будет проделано с нужным нам файлом. Кстати, еще неизвестно, с каким профилем загрузится система AutoCAD, а изменить профиль мы сможем только после полного запуска, когда уже, возможно, это будет поздно. Кроме того, еще неизвестно, надо ли передавать в качестве параметра `'AutoCAD.Application'`, `'AutoCAD.Application.15'` или `'AutoCAD.Application.16'`[1]. Если мы понадеемся, что достаточно просто всегда указывать 15 или 16, то будем разочарованы — это вероятный, но не обязательный вариант.

Но главный подвох заключается в том, что наше приложение должно уметь запускать и AutoCAD 2002 (R15), и AutoCAD 2004 (R16), а знает только одну версию! Действительно, и в проект, и состав пакетов Delphi может быть включен только один файл AutoCAD_TLB.pas. Даже если мы попытаемся сжульничать и создадим два разноименных файла, например AutoCAD_TLB15.pas и AutoCAD_TLB16.pas, то этот фокус не пройдет — не могут существовать два одноименных класса. В общем-то основные объекты и методы в R15 и R16 не отличаются по сути, но отличаются своими `GUID`, например в R15 для `IAcadApplication`

```
GUID = '{8E75D910-3D21-11D2-85C4-080009A0C626}',
```

а в R16 — `'{93BC4E71-AFE7-4AA7-BC07-F80ACDB672D5}'`.

Эти параметры будут "зашиты" в программу во время компиляции, поэтому добиться правильной работы разных версий AutoCAD невозможно.

В-третьих, вариант запуска системы AutoCAD может заключаться в использовании так называемого позднего связывания. В этом случае мы не должны импортировать библиотеку типов и должны работать "вслепую", обращаясь к свойствам и методам только на основе собственных знаний, проверить которые компилятор не сможет.

При таком варианте мы должны объявлять

```
Var AcadApplication: OleVariant;
```

и использовать для получения указателя на `AcadApplication` примерно такую функцию:

```
function ruGetOrCreateAcadAsOleObject(AcadAppString: string): OleVariant;
begin
  try
    Result := GetActiveOleObject(AcadAppString);
  except
    try
      Result := CreateOleObject(AcadAppString);
    except
     ShowMessage ('Не удалось запустить AutoCAD');
    end;
  end;
end;
```

[1] А для AutoCAD 2005 придется указывать `'AutoCAD.Application.16.1'`.

А как же быть с AcadAppString? Какую строку передавать, мы можем определить с помощью специальной функции (листинг 20.3). Единственное, что мы знаем точно — это полное имя файла acad.exe. Вот от него и будем выполнять поиск.

Листинг 20.3. Функция ruFindAcadAppString

```
function ruFindAcadAppString(ExeName: string): string;
{
Локальная функция поиска для заданного EXE-файла
}
  function FindForAppVer(AcadApp, ExeName: string): string;
  var
    Acad_CLSID, AcadLocalServer32: string;
  begin
    Result := '';
    Acad_CLSID :='';
    AcadLocalServer32:='';
    try
{ Пытаемся прочитать из реестра параметр "по умолчанию", например в ветви
[HKEY_LOCAL_MACHINE\SOFTWARE\Classes\AutoCAD.Application\CLSID]
@="{8E75D911-3D21-11d2-85C4-080009A0C626}", где параметр по умолчанию обозначен
символом @, а в функции RegReadString (модуль JclRegistry из библиотеки JEDI Code
Library) обозначен ''
}
    Acad_CLSID:= RegReadString(HKEY_CLASSES_ROOT, AcadApp + '\CLSID',
 '');
    except
    end;
{Если такая ветвь существует и CLSID найден, ищем в ветви
[HKEY_CLASSES_ROOT\CLSID\{8E75D911-3D21-11d2-85C4-080009A0C626}\LocalServer32],
в которой может быть указано полное имя acad.exe
@="c:\\Acad\\2002\\acad.exe /Automation"
}
    if (Acad_CLSID <> '') then begin
      try
      AcadLocalServer32:= RegReadString(HKEY_CLASSES_ROOT,
                  'CLSID\'+Acad_CLSID+'\LocalServer32','');
      except
      end;
      if  (AcadLocalServer32 <> '') then
{Проверяем, "наш" ли acad.exe закреплен за этим идентификатором
}
        if StrFind(ExeName, AcadLocalServer32, 1) > 0 then
          Result := AcadApp;
     end;
  end;
{Основная функция}
begin
{Перебираем три возможных варианта}
  Result := FindForAppVer('AutoCAD.Application', ExeName);
```

```
  if Result = '' then
    Result := FindForAppVer('AutoCAD.Application.15', ExeName);
  if Result = '' then
    Result := FindForAppVer('AutoCAD.Application.16', ExeName)
end;
```

Теперь мы можем быть уверены, что получили правильное наименование OLE-объекта, или, если получим пустую строку, будем точно знать, что такая система AutoCAD не зарегистрирована в качестве COM-сервера.

Теперь мы можем запускать AutoCAD любым способом. Многочисленные эксперименты показали, что оптимальным является такой алгоритм:

1. Поиск запущенной системы AutoCAD, причем "нашей" версии. Такая система AutoCAD, например, может быть запущена и не из нашего стартера или уже запущена из стартера. Если обнаружена *подходящая* работающая система AutoCAD, то ей через COM-интерфейс посылается указание загрузить заданный файл.
2. Если подходящая работающая система AutoCAD не найдена, то производится запуск "нашей" системы AutoCAD с передачей посредством командной строки всех требуемых параметров.

Такая схема работы позволяет, запустив систему AutoCAD из стартера (или подключившись к уже работающей), при необходимости отправлять в нее все новые файлы. Конечно, файл можно открыть и из самой системы AutoCAD, но иногда из стартера это делать удобнее и быстрее, т. к. в нем имеется независимый список последних файлов, да еще комментированный. Единственное ограничение, на которое мы пойдем, — это запрет изменения версии уже работающей системы AutoCAD. Сделано это специально, т. к. в реальной работе это никому не нужно и может служить только развлечением для "обезьяны с гранатой".

Проблемы с файлом автозагрузки

После загрузки файла должны быть загружены наши библиотеки. Загрузка библиотек и инициализация системы предусматривается во время загрузки файла acaddoc.lsp. Файл с таким именем автоматически загружается в каждый документ, но для этого он должен находиться на путях поиска AutoCAD. Перечень каталогов, в которых AutoCAD ищет файлы поддержки, задается в профиле, следовательно, в момент загрузки или создания нового рисунка в AutoCAD должен быть установлен наш текущий профиль системы ruCAD.

При запуске системы AutoCAD в виде процесса в командной строке мы можем указать профиль и все произойдет, как надо. Но AutoCAD запоминает последний использованный профиль и, если в командной строке нет соответствующего параметра, загружается именно с этим профилем. Мы вознамерились подключаться и к уже работающей системе AutoCAD и должны учитывать, что в ней может быть установлен иной профиль. Но даже если мы отправим в AutoCAD указание об установке профиля ruCAD, файл acaddoc.lsp может не загружаться автоматически — есть такая неприятная особенность работы объектной модели. Придется посылать команду принудительной загрузки acaddoc.lsp.

Возможна и противоположная ситуация — загрузка посторонних автозагрузочных файлов. У пытливых пользователей их могут оказаться десятки — в виде "отходов

жизнедеятельности" различных, иногда давно уже ликвидированных, программ. Мы сделаем так, чтобы после завершения работы ruCAD файлы автозагрузки удалялись и не мешали работе других приложений.

Файл acaddoc.lsp будет создаваться из шаблона (листинг 20.4). В файле шаблона используются выделенные полужирным шрифтом обозначения переменных, вместо которых подставляются соответствующие значения.

Листинг 20.4. Шаблон файла acaddoc.lsp

```
(vl-load-com)
(setq *ru_root_dir* "%RU_ROOT_DIR%")
(load (strcat *ru_root_dir* "%RU_MAIN_FAS_LIB%"))
(if (< (atoi "%RU_ACAD_VERSION%") 16)
 (progn
  (if (not (member "ru_MainLib.arx" (arx)))
    (arxload (findfile "ru_MainLib.arx")))
  (setq *ru_use_ru_arx* T)
)
  (setq *ru_use_ru_arx* nil)
)
;;; Загрузка ExpressTools
(ru-express-load)
;;; Загрузка DOSLib
(if (or (ru-doslib-load) *ru_use_ru_arx*)
 (progn
   (setq   *ru_current_user_info*
    (list
     (cons "LOGIN"  "%RU_USER_LOGIN%")
     (cons "PASSWORD"  "%RU_USER_PASSWORD%")
     (cons "TITLE"  "%RU_USER_TITLE%")
     (cons "LONG_NAME"  "%RU_USER_LONG_NAME%")
     (cons "WORK_GROUP"  "%RU_USER_WORK_GROUP%")
     (cons "WORK_DIR"  "%RU_USER_WORK_DIR%")
     (cons "BIT_RIGHTS" %RU_USER_BIT_RIGHTS%)
   )
  )
   (ru-init-start-rucad)
)
 (alert "Система не сможет работать. Не загружена ни MainLib, ни DosLib")
)
```

Реализация программы-стартера

После основательной теоретической подготовки мы можем написать программу. Не будем тратить место на исходный текст формы, а более подробно прокомментируем текст модуля (листинг 20.5). Не имеющие принципиального значения фрагменты исходного текста пропущены, а участки кода, на которые нужно обратить внимание, выделены полужирным шрифтом.

Листинг 20.5. Файл frmStarter.pas

```
unit frmStarter;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  Dialogs, Menus, ComCtrls, ExtCtrls, IniFiles, StdCtrls, ImgList,
  ActnList, JvComponent, JvMRUList, JvPlacemnt, JvSysComp;

type
  TFrmStart = class(TForm)
    StatusBar: TStatusBar;
    menuMainBar: TMainMenu;
...
    Images: TImageList;
    Acts: TActionList;
    aRunAcad: TAction;
    aReRegister: TAction;
    aExit: TAction;
    aAppList: TAction;
    aSelectAcad: TAction;
    aTips: TAction;
    aAbout: TAction;
    aAppMenuEdit: TAction;
    aRunApp: TAction;
    aFileNew: TAction;
    aUserSetup: TAction;
    JvMRUManager1: TJvMRUManager;
    JvCreateProcess: TJvCreateProcess;
    JvFormPlacement1: TJvFormPlacement;
    Image1: TImage;
    lbUserName: TLabel;
    lbUserS: TLabel;
    lbUserDir: TLabel;
    lbUserGroup: TLabel;
    LabelRuVersion: TLabel;
    LabelIsAdmin: TLabel;
    LabelAcadProduct: TLabel;
    LabelAcadVersion: TLabel;
    LabelAcadRelease: TLabel;
    LabelAcadProductRelease: TLabel;
    LabelAcadRun: TLabel;
    procedure FormCreate(Sender: TObject);
    procedure FormShow(Sender: TObject);
    procedure aSelectAcadExecute(Sender: TObject);
    procedure AppEventsHint(Sender: TObject);
    procedure aAboutExecute(Sender: TObject);
    procedure LoginDialog(FirstLogin: boolean);
    procedure aAppMenuEditExecute(Sender: TObject);
```

```
    procedure aRunAppExecute(Sender: TObject);
    procedure aRunAcadExecute(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure JvMRUManager1Click(Sender: TObject; const RecentName,
      Caption: string; UserData: Integer);
    procedure JvMRUManager1GetItemData(Sender: TObject;
      var Caption: string; var ShortCut: TShortCut; UserData: Integer);
    procedure aFileNewExecute(Sender: TObject);
    procedure FormCloseQuery(Sender: TObject; var CanClose: Boolean);
    procedure aExitExecute(Sender: TObject);
    procedure aTipsExecute(Sender: TObject);
    procedure aReRegisterExecute(Sender: TObject);
    procedure aUserSetupExecute(Sender: TObject);
  private
  protected
    procedure ShowUserInfo;
    procedure ShowAcadInfo;
    function MakeAcadDocLsp: boolean;
    function RunAcad(dwgname: string): boolean;
  public
    { Public declarations }
    AcadRun, IsDeveloper, IsUserAdmin: boolean;
    AcadDocLsp, AcadAppString, DwgComment, StorageIniFile, AcadRelease,
    AcadVersion, AcadProductName, AcadExeName, Login, Title, LongName,
    Password, WorkGroup, WorkDir: WideString;
    BitRights: Integer;
  end;

var
  FrmStart: TFrmStart;

implementation

uses
  ComObj, ActiveX, ComServ, JclLogic, JclSysInfo, ruUtils, ruAcad,
  ruFileUtils, ruTreeXMLUtils, ruUsersUtils, ruTipsSrv_TLB,
  ruUserLoginSvr_TLB, ruEditUsersSvr_TLB, ruXmlMenuSrv_TLB,
  ruShellFileDlgSvr_TLB, ruSplash, ruWebBrv, frmSelectAcad;

{$R *.DFM}

var
{------------------------
Объявления интерфейсов объектов, к которым мы будем обращаться через
COM
------------------------}
  LoginDlg: ILoginDlg;
  EditUsersDlg: IEditUsers;
  TipsOfDay: IruTipsOfDay;
  ruXmlTree: IXmlTree;
  FileDialog: IFileDialog;
  AcadApplication: OleVariant;
```

```
procedure TFrmStart.FormCreate(Sender: TObject);
begin
{-----------------------
Создание COM-объектов
-----------------------}
  LoginDlg := CreateComObject(CLASS_LoginDlg) as ILoginDlg;
  EditUsersDlg := CreateComObject(CLASS_EditUsers) as IEditUsers;
  FileDialog := CreateComObject(CLASS_FileDialog) as IFileDialog;
{-----------------------
Регистрация пользователя
-----------------------}
  LoginDialog(True);
  StorageIniFile := ruGet_LocalAppDataDir + 'Starter\starter.ini';
  AcadRun := False;
end;

procedure TFrmStart.LoginDialog(FirstLogin: boolean);
var
  FLogin, FTitle, FLongName, FPassword, FWorkGroup, FWorkDir: WideString;
  FBitRights: Integer;
  IniFileName: string;
  Ini: TIniFile;
begin
  if FirstLogin then
  begin
    IniFileName := ruIncludeTrailingBackslash(ruGet_LocalAppDataDir) +
      UserIniFileName;
    Ini := TIniFile.Create(IniFileName);
    FLogin := Ini.ReadString(UserIniLastParamsSection,
        UserIniLoginVarName, 'GUEST');
    Ini.Free;
    FLongName := GuestLongName;
    FPassword := 'BAD_PASSWORD';
    FWorkDir := ruGet_CurrentUserDir + 'Guest';
    FTitle := GuestTitle;
    FWorkGroup := GuestWorkGroup;
    FBitRights := 0;
  end else begin
    FLogin := Login;
    FLongName := LongName;
    FPassword := Password;
    FWorkDir := WorkDir;
    FTitle := Title;
    FWorkGroup := WorkGroup;
    FBitRights := BitRights;
  end;
{-----------------------
Процедура регистрации
-----------------------}

  FBitRights := LoginDlg.Execute(FLogin, FTitle, FLongName,
    FPassword, FWorkGroup, FWorkDir);
```

```
{-----------------------
Если регистрация не выполнена в первый раз, программа завершается. При последующих
возможных перерегистрациях программа продолжает работать
-----------------------}

  if (FBitRights = 0) and FirstLogin then
  begin
    Hide;
    Application.Terminate;
    Exit;
  end;
  if (FBitRights = 0) then Exit;
  Login := FLogin;
  LongName := FLongName;
  Password := FPassword;
  WorkDir := FWorkDir;
  Title := FTitle;
  WorkGroup := FWorkGroup;
  BitRights := FBitRights;
  IsUserAdmin := TestBits(BitRights, AdminBit);
  IsDeveloper := TestBits(BitRights, DeveloperBit);
  ShowUserInfo;
end;

procedure TFrmStart.ShowUserInfo;
{-----------------------
Отображение свойств пользователя на форме и установка доступа к действиям с
ограниченным доступом
-----------------------}

  function MakeAcadCurrentUserDir(FWorkDir: string): string;
  begin
    Result := FWorkDir;
{-----------------------
Если в реестре у пользователя не задан рабочий каталог, рабочим каталогом
назначается папка "Мои документы"
-----------------------}
    if FWorkDir = '' then Result := JclSysInfo.GetPersonalFolder;
  end;

begin

  lbUserName.Caption := Title + ' ' + LongName;
  WorkDir := MakeAcadCurrentUserDir(WorkDir);
  lbUserDir.Caption := WorkDir;
  lbUserGroup.Caption := WorkGroup;
  if IsUserAdmin then LabelIsAdmin.Caption := 'Да'
  else LabelIsAdmin.Caption := 'Нет';
  aSelectAcad.Enabled := IsUserAdmin and (not AcadRun);
  aAppMenuEdit.Enabled := IsUserAdmin;

end;
```

```
procedure TFrmStart.FormShow(Sender: TObject);
var
  Param_str: string;
Begin
// Отображение свойств AutoCAD
  ShowAcadInfo;
// Восстановление размещения формы
  JvFormPlacement1.IniFileName := StorageIniFile;
  JvFormPlacement1.IniSection := 'ruCAD-Starter';
  JvFormPlacement1.RestoreFormPlacement;
{-----------------------
Проверка параметра командной строки. Если параметром является имя DWG-файла, сразу
запускается AutoCAD
-----------------------}

  if (ParamCount > 0) then
  begin
    Param_str := ruGetParameter;
    if (AnsiCompareText('.dwg', ExtractFileExt(Param_str)) = 0) then
      RunAcad(Param_str);
  end;
end;

procedure TFrmStart.ShowAcadInfo;
{-----------------------
Уточнение и отображение информации об AutoCAD
-----------------------}
begin
  AcadExeName := ruGet_AcadExeName;
  aRunAcad.Enabled := AcadExeName <> '';
  aFileNew.Enabled := aRunAcad.Enabled;
  AcadVersion := ruGet_AcadVersion;
  AcadRelease := ruGet_AcadProductRelease;
  AcadProductName := ruGet_AcadProductName;
  LabelAcadProduct.Caption := AcadProductName;
  LabelAcadProductRelease.Caption := AcadRelease;
  AcadAppString := ruFindAcadAppString(AcadExeName);
  AcadDocLsp := ruGet_LocalAcadAllVersionDir + AcadVersion +
    '\acaddoc.lsp';
  LabelRuVersion.Caption := 'Версия: "' + ruGet_RuCADVersion +
'" (для AutoCAD R' + AcadVersion + ')';
  LabelAcadVersion.Caption := 'R' + AcadVersion +
                              ' (' + AcadAppString + ')';
  aRunAcad.Enabled := (AcadExeName <> '') and (AcadAppString <> '');
  aFileNew.Enabled := aRunAcad.Enabled;
  JvMRUManager1.AutoEnable := aRunAcad.Enabled;
  if AcadRun then LabelAcadRun.Caption := 'Да'
  else  LabelAcadRun.Caption := 'Нет';
  if (AcadAppString = '') then LabelAcadRun.Caption :=
      'Недоступен для запуска, неизвестна версия AutoCAD.Application!';
 end;
```

```
procedure TFrmStart.aSelectAcadExecute(Sender: TObject);
begin
  if ruSelectAcad then
    ShowAcadInfo;
end;

procedure TFrmStart.aAboutExecute(Sender: TObject);
{-----------------------
В качестве диалогового окна "О программе" запускается мини-браузер с отображением
заданной страницы
-----------------------}
begin
  ruWebBrowse(IncludeTrailingBackslash(ruGet_HtmlDir) + 'about.html');
end;

procedure TFrmStart.aAppMenuEditExecute(Sender: TObject);
{-----------------------
Для редактирования меню дополнительных приложений запускаем тот же самый редактор,
который используем для редактирования XML-меню
-----------------------}

var
  CmdLine: string;
begin
  CmdLine := ruGet_RootDir + 'bin\ruXmlMenuEdit.exe "' + ruGet_LocalAppDataDir
+'Starter\ruStartApp.xml"';
{-----------------------
Для надежного запуска приложений используем специальный компонент
-----------------------}
  JvCreateProcess.CommandLine := CmdLine;
  JvCreateProcess.Run;
end;

procedure TFrmStart.aRunAppExecute(Sender: TObject);
var
  ExeFile, Dir, Param_str, Name, XmlRootDir, XmlImagesDir, XmlFName,
    ExeFileAttrib, DirAttrib, ParamStrAttrib, ResultSection, SelectSection,
LastSelectedVar, StorageFname, ResultFName: string;
  ResDial: OleVariant;
  Ini: TIniFile;
  LastSelected: integer;
begin
  ExeFileAttrib := 'exe';
  DirAttrib := 'dir';
  ParamStrAttrib := 'paramstr';
  ResultSection := 'result';
  SelectSection := 'singleselect';
  LastSelectedVar := 'lastselected';

  XmlRootDir := ruGet_LocalAppDataDir;
  XmlFName := XmlRootDir + 'Starter\ruStartApp.xml';
  XmlImagesDir := XmlRootDir;
```

```
  StorageFname := XmlRootDir + 'Starter\starter.ini';
  ResultFName := XmlRootDir + 'Starter\result.ini';
  Ini := TIniFile.Create(ResultFName);
  LastSelected := Ini.ReadInteger(ResultSection, LastSelectedVar, 0);
  Ini.Free;
{-----------------------
Запуск внешних приложений, описанных в XML-меню, делаем точно так же и с помощью
того же COM-объекта, что и запуск LISP-программ
-----------------------}
  ruXmlTree := CreateComObject(CLASS_XmlTree) as IXmlTree;
  ResDial := ruXmlTree.Execute('Выбор приложения для запуска', XmlFName,
    XmlRootDir, XmlImagesDir, TreeViewedAttrib, IncludeAttrib,
    ImageAttrib, CommentAttrib, ExeFileAttrib, 'Выполнить', 'Отказ',
    StorageFname, ResultFName,'', False, IsDeveloper, False,
    LastSelected);
  if ResDial then
  begin
    Ini := TIniFile.Create(ResultFName);
    ExeFile := Ini.ReadString(SelectSection, ExeFileAttrib, '');
    Dir := Ini.ReadString(SelectSection, DirAttrib, '');
    Param_str := Ini.ReadString(SelectSection, ParamStrAttrib, '');
    Name := Ini.ReadString(SelectSection, TreeViewedAttrib, '');
    Ini.Free;
    if ExeFile <> '' then
    begin
      if dir = '%BIN%' then
        ExeFile := ruGet_RootDir + 'bin\' + ExeFile;
      if Param_str <> '' then Param_str := ' "' + Param_str + '"';
      JvCreateProcess.CommandLine := ExeFile + Param_str;
      JvCreateProcess.Run;
    end;
  end;
end;

procedure TFrmStart.aRunAcadExecute(Sender: TObject);
{-----------------------
Запуск AutoCAD
-----------------------}
var
  FileName: OleVariant;
  ResDial: OleVariant;
begin
  FileDialog.Create;
  with FileDialog do
  begin
    Caption := 'Выбор или создание DWG-файла';
    RootDir := WorkDir;
    StartDir := WorkDir;
    FilterMask := '*.dwg';
    DefaultFileName := 'new_drawing';
    CanEdit := True;
```

```
    CanCreateNewFile := True;
    MultiSelect := False;
// Вызов диалогового окна выбора файла
    ResDial := Execute;
  end;
  if ResDial then
  begin
    FileName := FileDialog.GetFile(0);
    RunAcad(FileName);
  end;
  FileDialog.Free;

end;
function TFrmStart.RunAcad(dwgname: string): boolean;
{-----------------------
Важнейшая функция, осуществляющая запуск AutoCAD
-----------------------}

  function FMakeAcadCommandLine(FAcadExeName, FAcadVersion,
   FDwgName: string): string;
{-----------------------
Локальная функция конструирования командной строки
-----------------------}

  var
     CfgName, TemplateName: string;
  begin
    CfgName := ruGet_LocalAcadAllVersionDir + FAcadVersion + '\ruCAD' +
      FacadVersion + 'arg';
    TemplateName := ruGet_LocalAcadAllVersionDir + FAcadVersion +
      '\Template\ACADISO';
    Result := '"' + FAcadExeName + '" ';
    if FileExists(FDwgName) then Result := Result + '"' + FDwgName + '" '
    else Result := Result + '/t "' + TemplateName + '" ';
    Result := Result + '/c "' + CfgName + '" ';
    Result := Result + '/p ruCAD  /nologo';
  end;

  function FOpenDwgInRunningAcad(FAcadAppString, FAcadVersion, FDwgName,
    FAcadDocLsp:string): boolean;
{-----------------------
Локальная функция загрузки файла в запущенной системе AutoCAD
-----------------------}
  var
    ProfileFile: string;
  begin
   Result := true;
   Try
// Пытаемся найти работающую систему AutoCAD
     AcadApplication := GetActiveOleObject(FAcadAppString);
// Если система AutoCAD найдена, то пытаемся установить профиль
    Try
```

```
// Имя ARG-файла с параметрами профиля
        ProfileFile := ruGet_LocalAcadAllVersionDir + FAcadVersion +
                '\ruCAD' + FAcadVersion + 'arg';
        AcadApplication.Preferences.Profiles.ImportProfile('ruCAD',
          ProfileFile, True);
      except
    // Ошибочная ситуация возникнет, если профиль уже импортирован
    // Сюда было вставлено отладочное сообщение
    //    ShowMessage ('Ошибка импорта профиля');
      end;
      try
// Пытаемся сделать текущим наш профиль
        AcadApplication.Preferences.Profiles.ActiveProfile := 'ruCAD';
      except
//     Ошибка возникнет, если профиль уже текущий
//      ShowMessage ('Сбой при профиле');
      end;
      if FileExists(DwgName) then
// Открываем заданный файл
        AcadApplication.Documents.Open(FDwgName, False)
      Else
// или создаем из своего шаблона
        AcadApplication.Documents.Add(ruGet_LocalAcadAllVersionDir +
         FAcadVersion + '\Template\ACADISO');
      try
// Принудительная загрузка acaddoc.lsp
AcadApplication.ActiveDocument.SendCommand(ruMakeLoadString
                                                  (FAcadDocLsp));
      except
        ShowMessage('Ошибка загрузки AcadDoc');
      end;
    except
      Result := False;
    end;
  end;

  function FRunAcadAsProcess(FAcadExeName, FAcadVersion, FDwgName,
                                      FWorkDir:string): boolean;
{-----------------------
Локальная функция запуска AutoCAD как процесса
-----------------------}

  begin
    Result := True;
    try
      if (FAcadExeName <> '') and (FileExists(FAcadExeName)) then
      begin
        JvCreateProcess.CommandLine := FMakeAcadCommandLine(FAcadExeName,
          FAcadVersion, FDwgName);
        JvCreateProcess.CurrentDirectory := FWorkDir;
        JvCreateProcess.Run;
      end;
```

```
    except
      Result := false;
    end;
  end;
{-----------------------
Основная функция
-----------------------}
var
  Msg: string;
begin
  Result := true;
  if not MakeAcadDocLsp then
  begin
    Application.MessageBox('Не могу создать AcadDoc.lsp! Запуск AutoCAD
невозможен!', pchar(Title + ' ' + LongName + '!'), MB_OK + MB_ICONHAND +
MB_DEFBUTTON1 + MB_APPLMODAL);
    Result := False;
    Exit;
  end;
  Msg := Title + ' ' + LongName + ', ждите, ' + #13#10;
  if DwgName = '' then Msg := Msg + 'создаю новый файл...'
  else begin
    DwgComment := ruReadDirinfoFileDescription(DwgName);
    Msg := Msg + 'загружаю файл' + #13#10 + DwgName + #13#10 + '(' +
      DwgComment + ')';
// Добавление в список последних файлов
    JvMRUManager1.Add(DwgName, 0);
  end;
  ShowSplash(Msg, True);
// Попытка найти работающую систему AutoCAD
  AcadRun := FOpenDwgInRunningAcad(AcadAppString, AcadVersion, DwgName,
    AcadDocLsp);
if not AcadRun then
  try
    if (AcadExeName <> '') and (FileExists(AcadExeName)) then
    begin
// Иначе запуск процесса
      AcadRun := FRunAcadAsProcess(AcadExeName, AcadVersion, DwgName,
      WorkDir);
    end;
  except
    Result := false;
  end;
  HideSplash;
  Application.Minimize;
  if AcadRun then LabelAcadRun.Caption := 'Да'
  else LabelAcadRun.Caption := 'Нет';
  aSelectAcad.Enabled := IsUserAdmin and (not AcadRun);
end;
```

```
function TFrmStart.MakeAcadDocLsp: boolean;
{-----------------------
Функция создания acaddoc.lsp
-----------------------}

var
  FRootDir, FWorkDir, FMainLibFas, TemplateName: string;
  AcadDocLspStrList: TStringList;

begin
  Result := False;
  TemplateName := ruGet_AllUsersDir + 'template\acaddoc.lsp';
  if not FileExists(TemplateName) then Exit;
  AcadDocLspStrList := TStringList.Create;
  AcadDocLspStrList.Clear;
  AcadDocLspStrList.LoadFromFile(TemplateName);
  {
  Допустимые шаблоны переменных:
  %RU_ROOT_DIR%           Пример: "c:\\.ru\\cad\\"
  %RU_MAIN_FAS_LIB%       Пример: "All Users\\app\\ru-lib-main.fas"
  %RU_ACAD_VERSION%       Пример: "15"

  %RU_USER_LOGIN%         Пример: "ADMIN"
  %RU_USER_PASSWORD%      Пример: "rucad"
  %RU_USER_TITLE%         Пример: "Товарищ"
  %RU_USER_LONG_NAME%     Пример: "Первый администратор"
  %RU_USER_WORK_GROUP%    Пример: "Water"
  %RU_USER_WORK_DIR%      Пример: "c:\\.ru\\cad\\samples\\dwg\\Water\\"
  %RU_USER_BIT_RIGHTS%    Пример: 4095
  Для LISP необходимо заменить одинарные слэши на двойные в переменных, содержащих
эти символы
}
  FRootDir := ruGet_RootDir;
  FRootDir := StringReplace(FRootDir, '\', '\\',
                [rfReplaceAll, rfIgnoreCase]);
  FWorkDir := WorkDir;
  FWorkDir := StringReplace(FWorkDir, '\', '\\',
                [rfReplaceAll, rfIgnoreCase]);
  FMainLibFas := 'All Users\\app\\ru-lib-main.fas';
  AcadDocLspStrList.Text := StringReplace(AcadDocLspStrList.Text,
    '%RU_ROOT_DIR%', FRootDir, [rfReplaceAll, rfIgnoreCase]);
  AcadDocLspStrList.Text := StringReplace(AcadDocLspStrList.Text,
    '%RU_MAIN_FAS_LIB%', FMainLibFas, [rfReplaceAll, rfIgnoreCase]);
  AcadDocLspStrList.Text := StringReplace(AcadDocLspStrList.Text,
    '%RU_ACAD_VERSION%', AcadVersion, [rfReplaceAll, rfIgnoreCase]);
  AcadDocLspStrList.Text := StringReplace(AcadDocLspStrList.Text,
    '%RU_USER_LOGIN%', Login, [rfReplaceAll, rfIgnoreCase]);
  AcadDocLspStrList.Text := StringReplace(AcadDocLspStrList.Text,
    '%RU_USER_PASSWORD%', Password, [rfReplaceAll, rfIgnoreCase]);
  AcadDocLspStrList.Text := StringReplace(AcadDocLspStrList.Text,
    '%RU_USER_TITLE%', Title, [rfReplaceAll, rfIgnoreCase]);
```

```
  AcadDocLspStrList.Text := StringReplace(AcadDocLspStrList.Text,
    '%RU_USER_LONG_NAME%', LongName, [rfReplaceAll, rfIgnoreCase]);
  AcadDocLspStrList.Text := StringReplace(AcadDocLspStrList.Text,
    '%RU_USER_WORK_GROUP%', WorkGroup, [rfReplaceAll, rfIgnoreCase]);
  AcadDocLspStrList.Text := StringReplace(AcadDocLspStrList.Text,
    '%RU_USER_WORK_DIR%', FWorkDir, [rfReplaceAll, rfIgnoreCase]);
  AcadDocLspStrList.Text := StringReplace(AcadDocLspStrList.Text,
    '%RU_USER_BIT_RIGHTS%', IntToStr(BitRights), [rfReplaceAll, rfIgnoreCase]);
  AcadDocLspStrList.SaveToFile(AcadDocLsp);
  AcadDocLspStrList.Free;
  Result := True;
end;

procedure TFrmStart.JvMRUManager1Click(Sender: TObject; const RecentName,
  Caption: string; UserData: Integer);
// Выбор из списка последних файлов
begin
  RunAcad(RecentName);
end;

procedure TFrmStart.JvMRUManager1GetItemData(Sender: TObject;
  var Caption: string; var ShortCut: TShortCut; UserData: Integer);
// Добавление в список последних файлов
var
  s: string;
begin
  s := ruReadDirinfoFileDescription(Caption);
  if s <> Caption then s := ' (' + s + ')'
  else s := '';
  Caption := Caption + s;
end;

procedure TFrmStart.aFileNewExecute(Sender: TObject);
// Запуск AutoCAD с созданием нового файла
begin
  RunAcad('');
end;

procedure TFrmStart.FormCloseQuery(Sender: TObject;
 var CanClose: Boolean);
{
Этот код отрабатывается при любой попытке закрытия формы приложения
Перед закрытием формы выдается запрос на согласие закрыть и AutoCAD
}
var
  S: string;
begin
  CanClose := True;
  if AcadRun then
  begin
    S := Title + ' ' + LongName + '!';
```

```
    try
      AcadApplication := GetActiveOleObject(AcadAppString);
      if
        Application.MessageBox('Действительно желаете выйти из ruCAD и закрыть
AutoCAD?', pchar(S), MB_YESNO + MB_ICONQUESTION + MB_DEFBUTTON2
        + MB_APPLMODAL) = ID_YES then
        AcadApplication.Quit
      else CanClose := False;
    except
    end;
  end;
end;

procedure TFrmStart.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  DeleteFile(AcadDocLsp);
  JvFormPlacement1.SaveFormPlacement;
end;

procedure TFrmStart.aExitExecute(Sender: TObject);
begin
  Close;
end;

procedure TFrmStart.aTipsExecute(Sender: TObject);
{
Вывод бесполезных полезных советов
}
var
  TipsFileName, TipsIniFileName: string;
begin
  TipsFileName := ruGet_LocalAppDataDir + 'TipsOfDay\ruCAD.tips';
  TipsIniFileName := ruGet_LocalAppDataDir + 'TipsOfDay\tips.ini';
  TipsOfDay := CreateComObject(CLASS_ruTipsOfDay) as IruTipsOfDay;
  TipsOfDay.ShowTips(TipsFileName, Title + ' ' + LongName,
    'Все ли Вы знаете о ruCAD?',
    TipsIniFileName, 'ruCAD' + '-' + Login, IsDeveloper,
    False, -1);
end;

procedure TFrmStart.aReRegisterExecute(Sender: TObject);
{
Перерегистрация пользователя
}
begin
  LoginDialog(False);
end;

procedure TFrmStart.aUserSetupExecute(Sender: TObject);
{
Редактирование пользовательских настроек
}
```

```
var
  FLogin, FTitle, FLongName, FPassword, FWorkGroup, FWorkDir: WideString;
  FBitRights: Integer;
begin
  FLogin := Login;
  FLongName := LongName;
  FPassword := Password;
  FWorkDir := WorkDir;
  FTitle := Title;
  FWorkGroup := WorkGroup;
  FBitRights := BitRights;
  FBitRights := EditUsersDlg.Edit(FLogin, FTitle, FLongName, FPassword,
    FWorkGroup, FWorkDir, FBitRights);
  if not IsUserAdmin then
  begin
    Login := FLogin;
    LongName := FLongName;
    Password := FPassword;
    WorkDir := FWorkDir;
    Title := FTitle;
    WorkGroup := FWorkGroup;
    BitRights := FBitRights;
    IsUserAdmin := TestBits(BitRights, AdminBit);
    IsDeveloper := TestBits(BitRights, DeveloperBit);
    ShowUserInfo;
  end;
end;
end.
```

Меню приложений

Внешние приложения, запускаемые из стартера, включаются в XML-меню, пример которого приведен в листинге 20.6.

Листинг 20.6. Пример XML-меню внешних приложений для стартера

```
<?xml version='1.0' encoding='windows-1251' ?>
<Программы>
  <group name='Программы ruCAD'>
    <app name='LISP-Explorer'
         dir='%BIN%'
         exe='ruLispExplorer'
         paramstr=''>Проводник по исходным текстам LISP-функций</app>
    <app name='Редактор XML'
         dir='%BIN%'
         exe='ruXmlMenuEdit.exe'
         paramstr=''>Редактор XML-меню</app>
    <Редактор_настроек name='Редактор настроек'
         dir='%BIN%'
```

```
          exe='ruXmlMenuEdit.exe'
          paramstr=
 'c:\.ru\cad\Local Settings\Application Data\ruCAD\Starter\ruReg.xml'>
          Редактор настроек, записываемых в реестр
    </Редактор_настроек>
  </group>
  <group name='Другие'>
    <app name='Блокнот'
        dir=''
       exe='c:\WINDOWS\notepad.exe'
         paramstr=''>Обычный блокнот (Notepad)</app>
  </group>
</Программы>
```

В дальнейшем программу-стартер мы будем дополнять новыми возможностями, вынесем в нее все настройки системы и прикладных программ (разумеется, с использованием любимых XML-меню), но в книге на этих вопросах больше задерживаться не будем.

Замечание

Значительные изменения придется вносить после разработки программы-инсталлятора и пробных запусков системы на посторонних компьютерах *(см. главу 36)*.

ГЛАВА 21

Рисование формата

В *главе 2* мы упоминали о необходимости разработки специальной программы рисования форматов чертежей. С рисования формата часто начинают осваивать AutoCAD. Обычно используют шаблоны с готовыми форматами, блоки форматов или просто применяют в качестве прототипа старые рисунки, иногда только ради подходящего формата. В *главе 4* мы уже писали, почему так делать мы не будем. Мы напишем универсальную программу, которая позволит нарисовать формат любого размера с любой формой основной надписи. Основные надписи мы сделаем в виде блоков с атрибутами, для того, чтобы графы основных надписей было бы удобно заполнять и редактировать и вручную, и программно. Формат мы будем рисовать в соответствии с ГОСТ 21.101-97. Инструменты для рисования форматов и основных надписей для машиностроительных чертежей легко получить модификацией нашей программы.

Программу мы разделим на две части:

- подготовка исходных данных, которая будет выполняться с помощью специального Мастера, написанного на Delphi;
- отрисовка формата по подготовленным данным, реализованная на языке LISP.

Диалоговое окно на Delphi мы, по сложившейся традиции, напишем в виде ActiveX DLL. Более мы не будем разжевывать процесс создания такого приложения, остановимся только на ключевых моментах. Рассматривать мы будем скриншоты готовой программы, они будут хорошими иллюстрациями к фрагментам кода.

Формирование окружения программы

Для того чтобы изолировать данные, используемые программой рисования формата, создаем специальный каталог %RuCadRootDir%\LocalSettings\Application Data\ruCAD \Format, в котором будут располагаться следующие файлы и папки:

- Формат.xml — справочник размеров листов;
- Empl.xml — справочник должностей и фамилий;
- DefaultObozn.xml — справочник стандартной структуры обозначения документов;
- DefaultFormat.ini — стандартные данные по умолчанию для новых документов;

- папка Blocks с файлами:
  - ru_format_base.dwg — блок основной надписи для чертежей зданий и сооружений;
  - ru_format_izd.dwg — блок основной надписи для чертежей изделий;
  - ru_format_text_first.dwg — блок основной надписи для первых листов текстовых документов;
  - ru_format_text_next.dwg — блок основной надписи для последующих листов текстовых документов;
  - ru_format_register.dwg — блок-граф инвентарного номера подлинника;
  - ru_format_sogl.dwg — блок-граф согласований.
- %RuCadRootDir%\Local Settings\Application Data\ruCAD\Format\Logo
  - logo.dwg — блок-логотип проектной организации (один из возможных).

Что находится в этих файлах, мы будем рассматривать далее.

Мастер рисования формата

Подготовку данных мы будем реализовывать в виде Мастера (Wizard). Это более подходящая для нашей цели форма приложения. В Мастере мы предусматриваем довольно много страниц с несложной логикой доступа.

Разработку приложения ведем по стандартной схеме:

1. Создаем автономный проект с формой Мастера и добиваемся его правильной работы.
2. Создаем проект ActiveX DLL, добавляем в него сервер автоматизации, редактируем библиотеку типов, редактируем модуль реализации, используя в нем отработанную форму.
3. Создаем тестовое Delphi-приложение для проверки работы сервера.
4. Пишем LISP-программу, вызывающую сервер и использующую результаты его работы.

Расписывать, как мы размещаем компоненты на форме и устанавливаем их свойства, у нас просто нет места. Рассмотрим приложение так, как его видит во время работы пользователь.

Работа начинается с установки размера листа (рис. 21.1). Основной компонент формы — `KWizard`[1]. Автор компонента — yu wei (так он себя называет). Это один из самых удачных Мастеров для Delphi. На пару с `KWizard` мы используем `KwizardRouteMapNodes`, обеспечивающий удобную навигацию по страницам Мастера (на форме слева). Автор этого компонента — Steve Forbes. Перелистывать страницы Мастера можно с помощью кнопок **Назад** и **Дальше** или щелчками по навигационной карте.

[1] **http://members.rogers.com/wyu66/index.htm**.

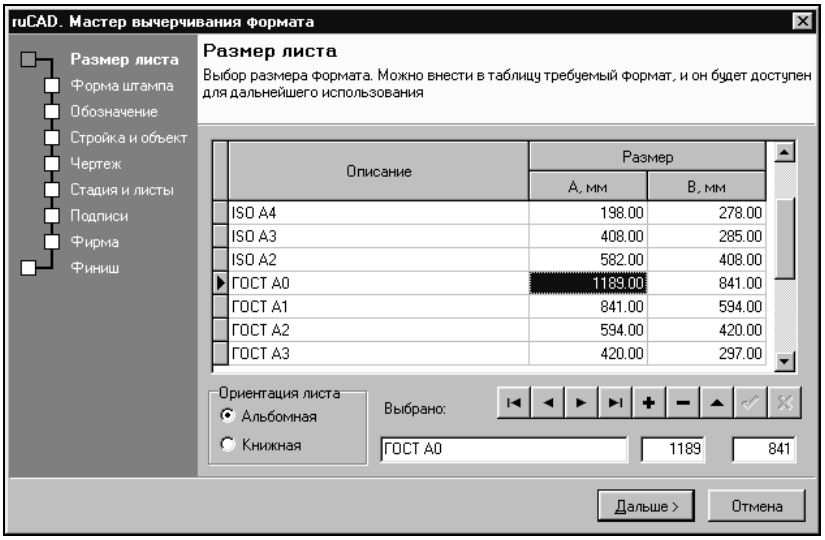


Рис. 21.1. Установка размера листа формата

Выбор размера листа

Для ведения небольшой базы данных форматов мы используем XML-таблицу, работа с которыми рассматривалась в *главе 16*. Отображение таблицы производится так же, как мы описывали ранее. Пользователь выделяет в таблице требуемый формат и выбирает ориентацию. Выбранные размеры отображаются в специальных компонентах `DbEdit` и затем используются для отрисовки. При необходимости новые форматы добавляются в таблицу. Разумеется, можно использовать любой нестандартный размер. Пример таблицы форматов (сокращенный) представлен в листинге 21.1.

Листинг 21.1. Файл Формат.xml

```
<?xml version='1.0' encoding='windows-1251' ?>
<table>
  <fields>
    <column_1 size='100' type='string'
         displayname='Описание' columnwidth='230'/>
    <column_2 size='9' type='float'
         displayname='Размер|A, мм' columnwidth='90'
         picklist='210|420|841' displayformat='0.00'/>
    <column_3 size='9' type='float'
         displayname='Размер|B, мм' columnwidth='90'
         picklist='297|594|1188' displayformat='0.00'/>
  </fields>
  <rows>
    <row>
      <column_1>ГОСТ A3</column_1>
```

```
      <column_2>420</column_2>
      <column_3>297</column_3>
    </row>
    <row>
      <column_1>Принтер рулонный CPF-136</column_1>
      <column_2>841</column_2>
      <column_3>340</column_3>
    </row>
</rows>
  <table/>
</table>
```

Так как в приложении используется несколько таблиц и сеток, была написана специальная процедура инициализации сетки (листинг 21.2). Она аналогична процедуре, рассмотренной в *главе 16*, но имеет универсальный характер. В дальнейшем ее можно вынести в отдельный модуль.

Листинг 21.2. Процедура загрузки XML с настройкой `DbGrid`

```
procedure TFormWizard.InitGrid(var XMLDataSet: TjanXMLDataSet2;
                               var XMLFileName: string;
                               var DBGrid: TDBGridEh;
                               var DataSourceXML: TDataSource;
                               MultiSelect:boolean);
var
  ColumnWidth, i, c: integer;
  DisplayName: string;
  Hidden: boolean;
  DisplayFormat, EditMask, PickListString: string;
  PickList: TStrings;
begin
  if FileExists(XMLFileName) then
  begin
    try
      XMLDataSet.close;
      XMLDataSet.XMLFile := XMLFileName;
      XMLDataSet.Open;
      c := XMLDataSet.FieldCount;
      DBGrid.DataSource := DataSourceXML;
      if MultiSelect then
        DBGrid.Options := DBGrid.Options + [dgMultiSelect];
      if c > 0 then
        for i := 0 to c - 1 do
        begin
          DisplayName := GetFieldAttribute(XMLDataSet,
            XMLDataSet.FieldDefs[i].Name,
            'displayname');
          if DisplayName <> '' then
            DBGrid.Columns[i].Title.Caption := DisplayName;
```

```
          EditMask := GetFieldAttribute(XMLDataSet,
            XMLDataSet.FieldDefs[i].Name,
            'editmask');
          if EditMask <> '' then
            DBGrid.Columns[i].EditMask := EditMask;
          DisplayFormat := GetFieldAttribute(XMLDataSet,
            XMLDataSet.FieldDefs[i].Name,
            'displayformat');
          if DisplayFormat <> '' then
            DBGrid.Columns[i].DisplayFormat := DisplayFormat;
          ColumnWidth :=
            strtointdef(GetFieldAttribute(XMLDataSet,
            XMLDataSet.FieldDefs[i].Name,
            'columnwidth'), 0);
          if ColumnWidth > 0 then
            DBGrid.Columns[i].Width := ColumnWidth;
            Hidden := strtobool(GetFieldAttribute(XMLDataSet,
            XMLDataSet.FieldDefs[i].Name,
            'hidden'));
          DBGrid.Columns[i].Visible := not Hidden;
          PickListString := GetFieldAttribute(XMLDataSet,
            XMLDataSet.FieldDefs[i].Name,
            'picklist');
          if PickListString <> '' then
          begin
            PickList := TStringList.Create;
            StringToList(PickListString, PickList, ['|']);
            DBGrid.Columns[i].PickList := PickList;
            PickList.Free;
          end;
        end;
    except
      Application.MessageBox(PChar('Не могу открыть Файл ' + XMLFileName),
        'Ошибка', MB_OK + MB_ICONHAND + MB_DEFBUTTON1 + MB_APPLMODAL);
    end;
  end
  else
    Application.MessageBox(PChar('Не найден ' + XMLFileName),
      'Ошибка', MB_OK + MB_ICONHAND + MB_DEFBUTTON1 + MB_APPLMODAL);
end;
```

Выбор формы основной надписи

В строительных чертежах используется четыре вида основных надписей. Выбор основной надписи производится на второй странице Мастера (рис. 21.2). В зависимости от вида основной надписи устанавливается доступ к другим страницам. Например, если выбрана форма чертежа изделия, то не будет доступен ввод наименований стройки и объекта, а при выборе формы продолжения текстовых документов будет доступен только ввод обозначения и номера листа.

Управление доступом к страницам осуществляется специальной процедурой (листинг 21.3).

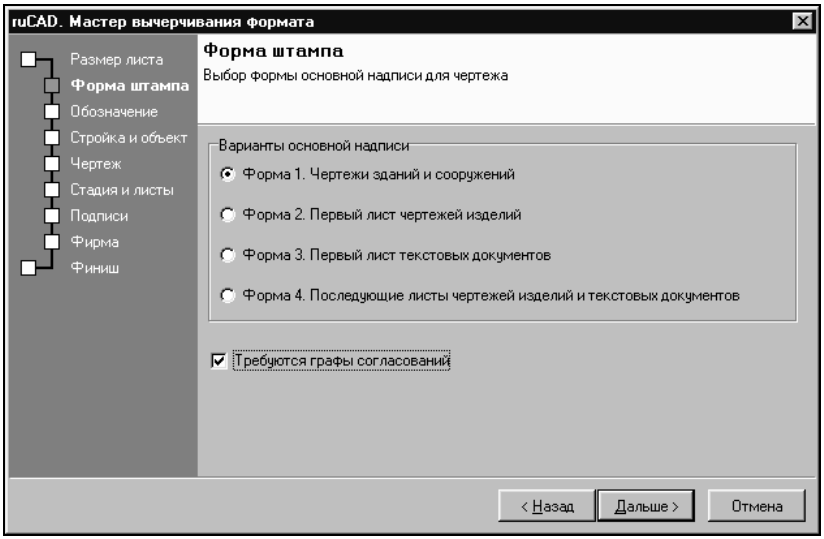


Рис. 21.2. Выбор формы основной надписи

Листинг 21.3. Управление доступом к страницам Мастера

```
procedure TFormWizard.ReEnablePages;
var
  IsIzd, IsSmall, IsBase, IsText: boolean;
begin
  IsBase := GroupBoxForm.ItemIndex = 0;
  IsIzd := GroupBoxForm.ItemIndex = 1;
  IsText := GroupBoxForm.ItemIndex = 2;
  IsSmall := GroupBoxForm.ItemIndex = 3;
  WizardPageBuildName.Enabled := IsBase;
  WizardPageDwgName.Enabled := IsBase or IsText;
  WizardPageNameIzd.Enabled := IsIzd;
  PanelStady.Visible := not IsSmall;
  PanelListAll.Visible := (not IsSmall) and (SpinEditList.Value = 1);
  WizardPageFamily.Enabled := not IsSmall;
  WizardPageFirm.Enabled := not IsSmall;
end;
```

Формирование обозначения документа

На следующем шаге формируется обозначение документа, указываемое в основных надписях. В соответствии с ГОСТ "в состав обозначения включают базовое обозначение, устанавливаемое по действующей в организации системе, и через дефис — марку и/или шифр раздела проекта". Если никакой *действующей системы* нет, можно просто написать в поле редактирования что угодно. Для организаций, придерживающихся определенных (иногда очень сложных) правил формирования базовых обозначений, в нашей программе предоставляется возможность формирования обозначения по стандартной структуре (рис. 21.3).

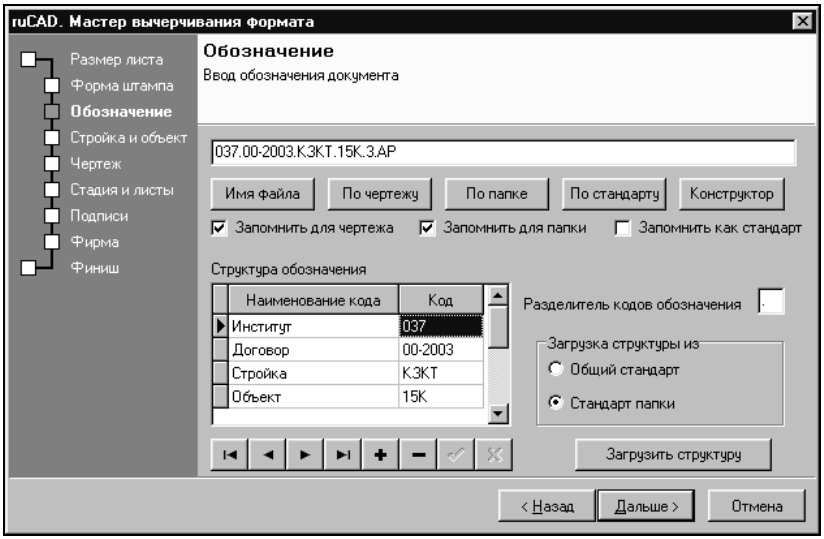


Рис. 21.3. Формирование обозначения документа

Структура обозначения может быть сформирована в специальной таблице (в ней также отображается XML-документ). Обозначение формируется последовательной конкатенацией кодов из таблицы с заданным разделителем кодов. Структура обозначения может быть загружена по общему стандарту или по стандарту папки. Для загрузки структуры обозначения необходимо установить требуемый переключатель (**Общий стандарт** или **Стандарт папки**), щелкнуть кнопку **Загрузить структуру**, отредактировать, при необходимости, структуру и щелкнуть кнопку **Конструктор**.

Обозначение документа и другие графы основной надписи удобно заполнять из ранее сохраненных образцов. На рис. 21.3 видны несколько кнопок, облегчающих заполнение обозначения:

- **Имя файла** — в поле редактирования подставляется имя текущего файла. Если имя файла создавалось по определенной системе *(см. главу 2)*, то оно само может служить обозначением или его заготовкой.
- **По чертежу** — обозначение загружается из файла имя_рисунка.ini. Запоминание обозначения в этом файле производится при выходе из Мастера, если установлен флажок **Запомнить для чертежа**.
- **По папке** — обозначение загружается из файла format.ini в папке текущего чертежа. Обычно это базовое обозначение для всех чертежей проекта. Запоминание обозначения в этом файле производится при выходе из Мастера, если установлен флажок **Запомнить для папки**.
- **По стандарту** — обозначение загружается из файла %RuCadRootDir%\LocalSettings \Application Data\ruCAD\Format\DefaultFormat.ini. Обычно это базовое обозначение для всех чертежей проектной организации. Запоминание обозначения в этом файле производится при выходе из Мастера, если установлен флажок **Запомнить как стандарт**.

По такой же схеме возможно сохранение и восстановление других граф основной надписи — наименований предприятий, объектов (рис. 21.4), чертежей (рис. 21.5).

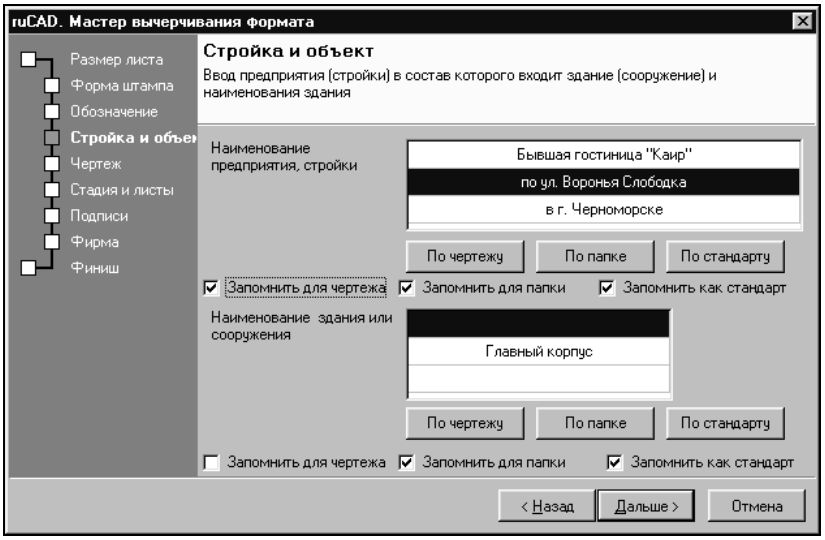


Рис. 21.4. Заполнение наименований стройки и объекта

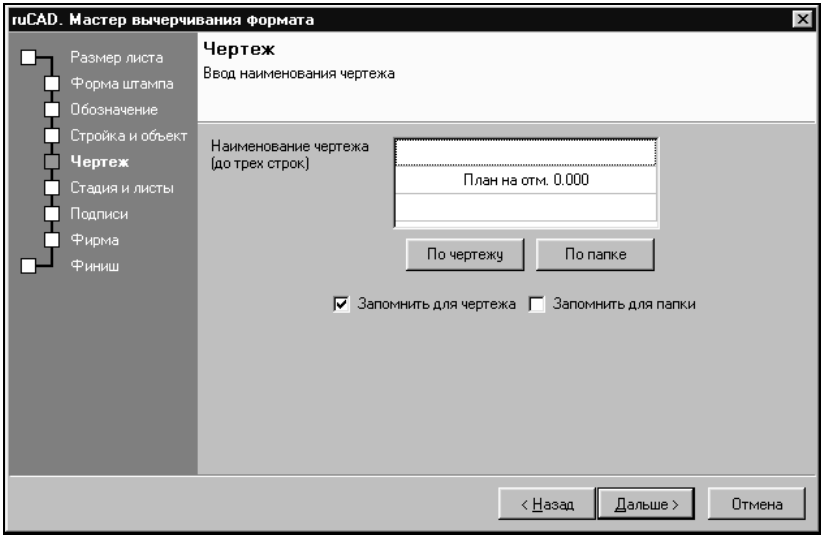


Рис. 21.5. Заполнение наименования чертежа

Сохранение и восстановление данных многострочных граф основной надписи

Во всех этих INI-файлах хранение данных производится единообразно, в секции `Format` (листинг 21.4).

Листинг 21.4. Пример файла format.ini

```
[Format]
;; Шесть переменных с наименованиями должностей
Engineer1=Директор
Engineer2=ГИП
Engineer3=Нач.отд
Engineer4=Ст.инж
Engineer5=Инж
Engineer6=НКонтр
;; Шесть переменных с фамилиями, соответствующими должностям
Family1=Фунт
Family2=Бендер
Family3=Козлевич
Family4=Брунс
Family5=Паниковский
Family6=Балаганов
;; Три переменных с наименованием проектной организации
Designer1=ОАО
Designer2=ГИПРОАВТОАГРЕГАТ
Designer3=г. Курган
;; Переменная с именем блока-логотипа проектной организации
LogoBlock=gaa.dwg
;; Обозначение документа
Obozn=045.000.001-1-АР.
;; Три переменных с наименованием стройки
Building1=Бывшая гостиница "Каир"
Building2=по ул. Воронья Слободка
Building3=в г. Черноморске
;; Три переменных с наименованием объекта
Object1=
Object2=Главный корпус
Object3=
;; Три переменных с наименованием чертежа
Drawing1=План на отм. 000
Drawing2=Разрезы
Drawing3=Фрагменты плана
;; Пять переменных с наименованием изделия
Izd1=
Izd2=Шкаф книжный
Izd3=системы
Izd4="Гей, славяне"
Izd5=
;; Три переменных с наименованием материала изделия
Material1=
Material2=Дуб березовый
Material3=
;; Номер листа
Page=1
;; Всего листов
AllPage=12
```

В файле DefaultFormat.ini имеется дополнительная секция `Blocks`, в которой указано, какие именно блоки используются для основных надписей разных типов

```
[Blocks]
;; Основная надпись чертежей зданий и сооружений
StampBlock0=ru_format_base
;; Основная надпись чертежей изделий
StampBlock1=ru_format_izd
;; Основная надпись первых листов текстовых документов
StampBlock2=ru_format_text_first
;; Основная надпись последующих листов текстовых документов
StampBlock3=ru_format_text_next
;; Блок согласований
SoglBlock=ru_format_sogl
;; Блок для инвентарного номера чертежа
RegisterBlock=ru_format_register.dwg
```

Заполнение многострочных данных производится в компоненте `StringGrid`. Восстановление и сохранение данных в ячейках `StringGrid` осуществляется так, как указано в листинге 21.5. По такой схеме читаются данные с изменяющимся номером в конце имени переменной (Drawing1, Drawing2, Drawing3 и т. п.).

Листинг 21.5. Заполнение ячеек значениями переменных из INI-файла

```
{
Заполнение ячеек StringGrid значениями переменной с регулярным именем
}
procedure LoadStringGridRowOrColFromIni(var StringGrid: TJvStringgrid;
                                        IsAddNum, IsColumn: boolean;
                                  NumRowOrCol, First, Last: integer;
                                 IniFile, Section, VarName: string);
var
  Ini: TIniFile;
  i, num: integer;
  s: string;
begin
  s := '';
  try
    Ini := TIniFile.Create(IniFile);
    for i := First to Last do
    begin
      num := i;
      if IsAddNum then num := i + 1;
      s := Ini.ReadString(Section, VarName + IntToStr(num), '');
      if IsColumn then StringGrid.Cells[NumRowOrCol, i] := s
      else StringGrid.Cells[i, NumRowOrCol] := s;
    end;
    Ini.Free;
  except
    s := '';
```

```
    for i := First to Last do
    begin
      num := i;
      if IsAddNum then num := i + 1;
      if IsColumn then StringGrid.Cells[NumRowOrCol, i] := s
      else StringGrid.Cells[i, NumRowOrCol] := s;
    end;
  end;
end;
{
Сохранение ячеек StringGrid в переменной с регулярным именем
}

procedure SaveStringGridRowOrColToIni(var StringGrid: TJvStringgrid;
                                      IsAddNum, IsColumn: boolean;
                                  NumRowOrCol, First, Last: integer;
                                  IniFile, Section, VarName: string);
var
  Ini: TIniFile;
  i, num: integer;
  s: string;
begin
  Ini := TIniFile.Create(IniFile);
  for i := First to Last do
  begin
    if IsColumn then S := StringGrid.Cells[NumRowOrCol, i]
    else  S := StringGrid.Cells[i, NumRowOrCol];
    num := i;
    if IsAddNum then num := i + 1;
    Ini.WriteString(Section, VarName + IntToStr(num), s);
  end;
  Ini.Free;
end;
{
Заполнение ячеек сетки наименования стройки
}
procedure TFormWizard.LoadBuild(IniFileName: string);
begin
  LoadStringGridRowOrColFromIni(JvStringgridNameBuild, True, True,
        0, 0, 2, IniFileName, FormatSectionName, FormatBuildVarName);
end;
{
Сохранение ячеек сетки наименования стройки
}
procedure TFormWizard.SaveBuild(IniFileName: string);
begin
  SaveStringGridRowOrColToIni(JvStringgridNameBuild, True, True, 0, 0, 2,
    IniFileName, FormatSectionName, FormatBuildVarName);
end;
```

Таким же образом обрабатываются данные и по другим многострочным графам.

Заполнение стадии проектирования и количества листов

Реализация этого этапа проста — стадия проектирования выбирается из списка, количество листов заполняется или "накручивается" в компоненте `SpinEdit`. Особенность только в том, что графа "Листов" заполняется только на первом листе, и, как только в поле **Лист** будет цифра больше единицы, поле **Листов** становится невидимым (рис. 21.6 и соответствующий листинг 21.6). Невидимым, а не недоступным для того, чтобы установленное в нем число (не сбрасываемое в ноль) не смущало пользователя.

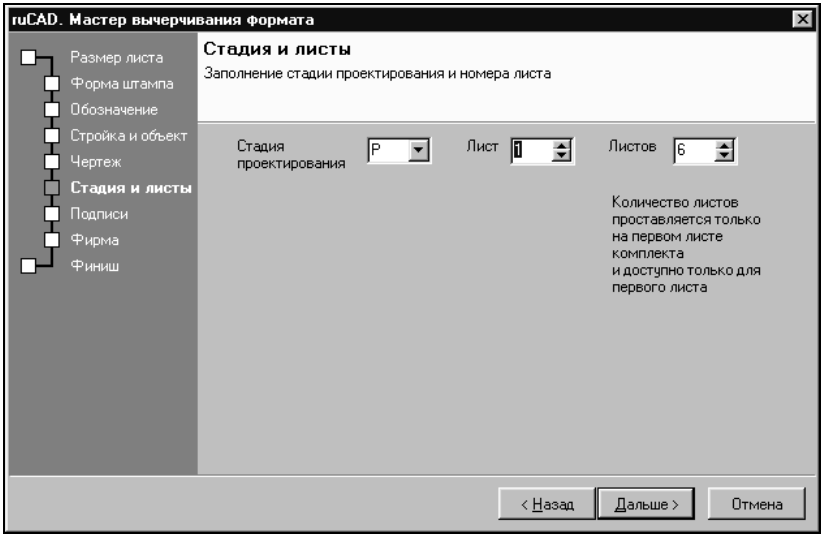


Рис. 21.6. Заполнение стадии проектирования и количества листов

Листинг 21.6. Управление вводом количества листов

```
procedure TFormWizard.ReEnableList;
var
  IsSmall: boolean;
begin
  IsSmall := GroupBoxForm.ItemIndex = 3;
  PanelStady.Visible := not IsSmall;
  PanelListAll.Visible := (not IsSmall) and (SpinEditList.Value = 1);
end;
```

Формирование набора подписей

Набор подписей для чертежей является условно-постоянным, но может изменяться для разных проектов. Для удобства предусмотрен справочник должностей и фами-

лий, естественно, также в формате XML. Справочник можно редактировать, перебрасывать из него требуемые пары "должность—фамилия" в состав подписей конкретного чертежа (рис. 21.7 и соответствующий листинг 21.7), сохранять и восстанавливать наборы подписей для чертежа, как стандарт.

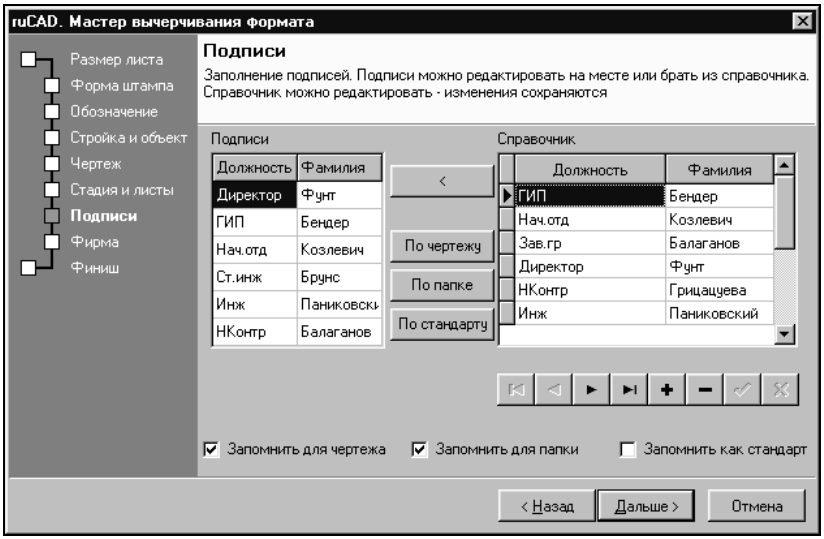


Рис. 21.7. Заполнение подписей

Листинг 21.7. Заполнение подписей из справочника

```
procedure TFormWizard.btnAddEmplClick(Sender: TObject);
var
  ARow: integer;
begin
  ARow := JvStringgridEmpl.Row;
  JvStringgridEmpl.Cells[0, ARow] := XMLDataSetEmpl.Fields[0].AsString;
  JvStringgridEmpl.Cells[1, ARow] := XMLDataSetEmpl.Fields[1].AsString;
end;
```

Формирование наименования организации

Наименование проектной организации также можно восстановить из различных источников. Это сделано с учетом того, что пользователь может работать под разными "крышами". Некоторые проектные фирмы используют вместо текстового названия организации логотип. Для применения логотипа необходимо подготовить его в виде DWG-файла, поместить в папку для логотипов. Блок логотипа можно выбрать при установке флажка **Использовать логотип**. Растровая миниатюра DWG-файла логотипа будет отображаться в Мастере (рис. 21.8 и соответствующий листинг 21.8).

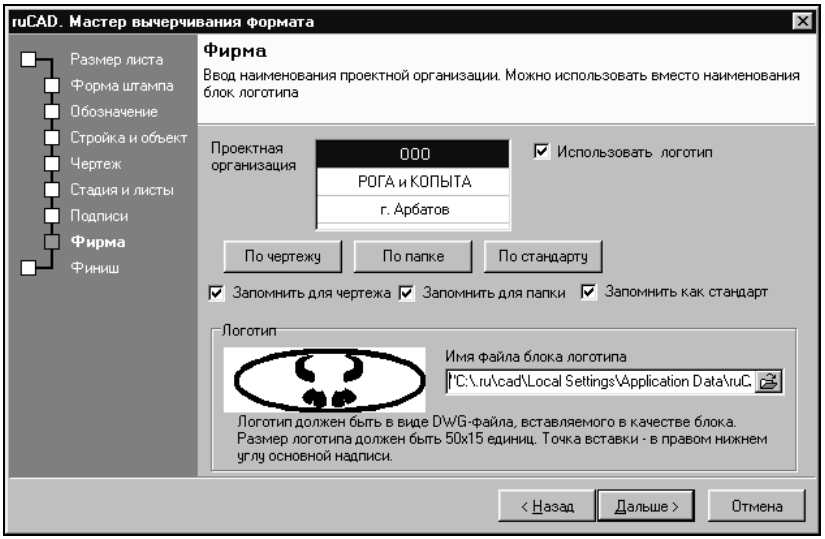


Рис. 21.8. Реквизиты проектной организации

Листинг 21.8. Отображение миниатюры логотипа

```
procedure TFormWizard.JvFilenameEditLogoAfterDialog(Sender: TObject;
  var Name: string; var Action: Boolean);
begin
  if Name <> '' then begin
    ruDwgPreviewLogo.Visible := True;
    ruDwgPreviewLogo.FileName := Name;
  end
  else begin
    ruDwgPreviewLogo.Visible := False;
    ruDwgPreviewLogo.FileName := '';
  end;
end;
```

Для отображения миниатюры предварительного просмотра, "зашитой" в DWG-файл, мы используем свой компонент `ruDwgPreview` (листинг 21.9).

Листинг 21.9. Компонент `ruDwgPreview`

```
unit ruDwgPreview;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
                                                  Dialogs, ExtCtrls;

type
  TruDwgPreview = class(TImage)
```

```
  private
    { Private declarations }
    FFileName: string;
    procedure SetFileName(Value: string);
    procedure ImportDwgThumbnail(DWGFileName: string);
  protected
    { Protected declarations }
  public
    { Public declarations }
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
  published
    { Published declarations }
    property FileName: string read FFileName write SetFileName;
  end;

procedure Register;

implementation

constructor TruDwgPreview.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
end;

destructor TruDwgPreview.Destroy;
begin
  inherited Destroy;
end;

procedure TruDwgPreview.SetFileName(Value: string);
begin
  FFileName := Value;
  ImportDwgThumbnail(pchar(Value));
end;

procedure TruDwgPreview.ImportDwgThumbnail(DWGFileName: string);
const
  ImageSentinel: array[0..15] of Byte =
  ($1F, $25, $6D, $07, $D4, $36, $28, $28, $9D, $57, $CA, $3F, $9D,
                                                    $44, $10, $2B);
type
  TDwgFileHeader = packed record
    Signature: array[0..5] of Char;
    Unused: array[0..6] of Char;
    ImageSeek: LongInt;
  end;
var
  DwgFile: file;
  StoreFileMode: Byte;
  DwgFileHeader: TDwgFileHeader;
  DwgSentinelData: array[0..15] of Byte;
```

```
function LoadBMPData(const BitmapInfo: PBitmapInfo): Boolean;
var
  BitmapHandle: HBITMAP;
  Bits: Pointer;
  NumColors: Integer;
  DC: HDC;

  function GetDInColors(BitCount: Word): Integer;
  begin
    case BitCount of
      1, 4, 8: Result := 1 shl BitCount;
    else
      Result := 0;
    end;
  end;

begin
  Result := False;
  DC := GetDC(0);
  if DC = 0 then
    Exit;
  try
    with BitmapInfo^ do
    begin
      NumColors := GetDInColors(bmiHeader.biBitCount);
      Bits := Pointer(Longint(BitmapInfo) + SizeOf(bmiHeader) +
                                                    NumColors *
        SizeOf(TRGBQuad));
    end;
    BitmapHandle := CreateDIBitmap(DC, BitmapInfo.bmiHeader, CBM_INIT,
                                   Bits, BitmapInfo^, DIB_RGB_COLORS);
    if BitmapHandle <> 0 then
    begin
       inherited Picture.Bitmap.Handle := BitmapHandle;
       inherited Show;
      Result := True;
    end;
  finally
    ReleaseDC(0, DC);
  end;
end;

procedure ProcessImageData;
type
  TImageDataHeader = packed record
    TotalCount: LongInt;
    ImagesPresent: Byte;
  end;
  TImageDataRecord = packed record
    DataType: Byte;
    StartOfData: LongInt;
    SizeOfData: LongInt;
  end;
```

```
  var
    ImageHeader: TImageDataHeader;
    ImageRecord: TImageDataRecord;
    BMPData, WMFData: TImageDataRecord;
    ThumbData: Pointer;
  begin
    BlockRead(DwgFile, ImageHeader, SizeOf(ImageHeader));
    if ImageHeader.TotalCount + FilePos(DwgFile) > FileSize(DwgFile) then
      Exit;
    FillChar(BMPData, SizeOf(BMPData), 0);
    FillChar(WMFData, SizeOf(WMFData), 0);
    while (IOResult = 0) and (ImageHeader.ImagesPresent > 0) do
    begin
      BlockRead(DwgFile, ImageRecord, SizeOf(ImageRecord));
      if (IOResult <> 0) or (ImageRecord.StartOfData > FileSize(DwgFile))
        then Break;
      case ImageRecord.DataType of
        2: BMPData := ImageRecord;
end;
      Dec(ImageHeader.ImagesPresent);
    end;
    if BMPData.StartOfData > 0 then
      ImageRecord := BMPData
else
      Exit;
    Seek(DwgFile, ImageRecord.StartOfData);
    GetMem(ThumbData, ImageRecord.SizeOfData);
    BlockRead(DwgFile, ThumbData^, ImageRecord.SizeOfData);
    try
     LoadBMPData(ThumbData);
    finally
      FreeMem(ThumbData);
    end;
  end;

begin
  Visible:=False;
  StoreFileMode := FileMode;
  FileMode := 0;
  System.Assign(DwgFile, DWGFileName);
  Reset(DwgFile, 1);
  FileMode := StoreFileMode;
  if IOResult <> 0 then
    Exit;
  try
    BlockRead(DwgFile, DwgFileHeader, SizeOf(DwgFileHeader));
    if (IOResult = 0) and (Copy(DwgFileHeader.Signature, 1, 4) = 'AC10')
       and
      (DwgFileHeader.ImageSeek <= FileSize(DwgFile)) then
    begin
      Seek(DwgFile, DwgFileHeader.ImageSeek);
      BlockRead(DwgFile, DwgSentinelData, SizeOf(DwgSentinelData));
```

```
      if (IOResult = 0) and CompareMem(@DwgSentinelData, @ImageSentinel,
        SizeOf(DwgSentinelData)) then
        ProcessImageData;
    end;
  finally
    Close(DwgFile);
  end;
end;

procedure Register;
begin
  RegisterComponents('ruCAD', [TruDwgPreview]);
end;

end.
```

Ввод граф для чертежей изделий

Чертежи изделий имеют особый набор граф, вынесенных на отдельную страницу Мастера (рис. 21.9). Эта страница доступна только при выборе соответствующей формы.

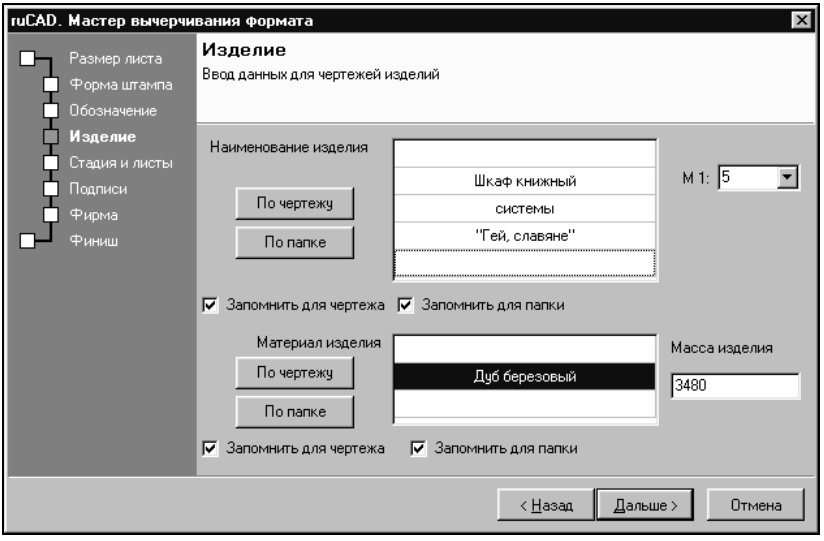


Рис. 21.9. Вариант основной надписи для строительного изделия

Завершение работы Мастера

На финишной странице Мастера (рис. 21.10) выводятся все подготовленные данные. Просмотрев их, пользователь может вернуться к требуемой странице и что-то изменить. При щелчке по кнопке **Отмена** работа Мастера завершается с результатом `ModalResult=mrCancel`, при щелчке по кнопке **Чертить** формируется файл результатов,

запоминаются все настройки, Мастер закрывается и возвращается результат `ModalResult=mrOk` (листинг 21.10).

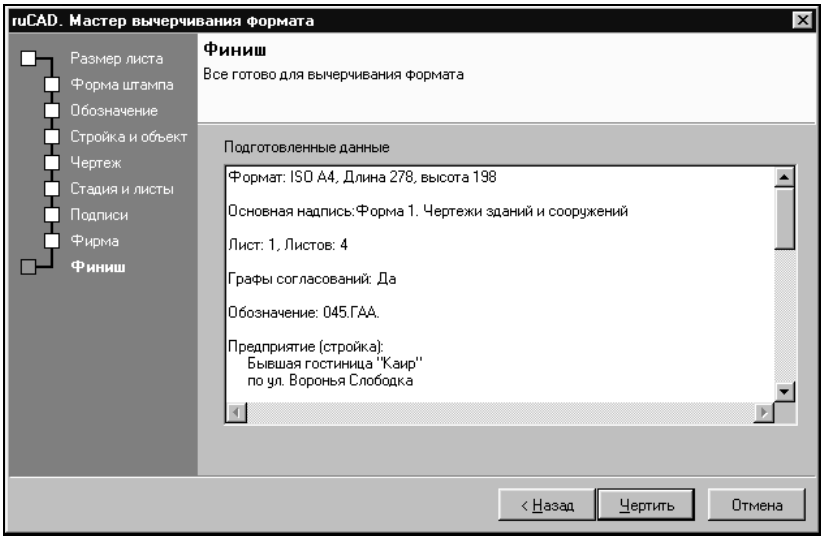


Рис. 21.10. Финишная страница Мастера

Листинг 21.10. Формирование результатов

```
procedure TFormWizard.MakeResult;
var
  BoolS: string;
begin
  BoolS := '0';
  SaveStringToIni(ResultIniFile, FormatSectionName,
    FormatHeightVarName, IntToStr(FormatHeight));
  SaveStringToIni(ResultIniFile, FormatSectionName,
    FormatWidthVarName, IntToStr(FormatWidth));
  SaveStringToIni(ResultIniFile, FormatSectionName,
    FormatOboznVarName, EditObozn.Text);
  SaveBuild(ResultIniFile);
  SaveObject(ResultIniFile);
  SaveDwg(ResultIniFile);
  SaveIzd(ResultIniFile);
  SaveMatIzd(ResultIniFile);
  SaveEmpl(ResultIniFile);
  SaveFirm(ResultIniFile);
  if CheckBoxSoglasov.Checked then
    BoolS := '1';
  SaveStringToIni(ResultIniFile, FormatSectionName,
    FormatSoglVarName, BoolS);
```

```
  SaveStringToIni(ResultIniFile, FormatSectionName,
    FormatFormVarName, IntToStr(GroupBoxForm.ItemIndex));
  SaveStringToIni(ResultIniFile, FormatSectionName,
    FormatScaleVarName, ComboBoxScale.Text);
  SaveStringToIni(ResultIniFile, FormatSectionName,
    FormatMassVarName, EditMass.Text);
  SaveStringToIni(ResultIniFile, FormatSectionName,
    FormatStadyVarName, ComboBoxStady.Text);
  SaveStringToIni(ResultIniFile, FormatSectionName,
    FormatListVarName, SpinEditList.Text);
  SaveStringToIni(ResultIniFile, FormatSectionName,
    FormatListAllVarName, SpinEditListAll.Text);
end;
```

Реализация COM-сервера

Мы уже знаем, что при редактировании библиотеки типов создается файл *_TLB.pas. Анализируя этот файл, программист (или сама Delphi) узнает о свойствах и методах данного объекта. Просматривая файл ruFormatWizardSrv_TLB.pas (листинг 21.11), мы видим, что объявлен единственный метод `Execute` с параметрами:

- `DwgFile` — полное имя файла рисунка, в котором будет нарисован формат;
- `ResIniFile` — полное имя INI-файла, в который следует записать результаты работы Мастера;
- `DefDir` — имя каталога, в котором находятся служебные файлы настроек по умолчанию;
- `TemplDir` — имя каталога, в котором находятся шаблоны для служебных таблиц.

Листинг 21.11. Фрагмент файла ruFormatWizardSrv_TLB.pas

```
unit ruFormatWizardSrv_TLB;

interface

uses Windows, ActiveX, Classes, Graphics, OleServer, OleCtrls, StdVCL;

const
  ruFormatWizardSrvMajorVersion = 1;
  ruFormatWizardSrvMinorVersion = 0;

  LIBID_ruFormatWizardSrv: TGUID =
'{0AAEE701-DAFB-11D7-8CF1-992A98189A54}';

  IID_IFormatWizard: TGUID = '{0AAEE702-DAFB-11D7-8CF1-992A98189A54}';
  CLASS_FormatWizard: TGUID = '{0AAEE704-DAFB-11D7-8CF1-992A98189A54}';
type

IFormatWizard = interface;
  IFormatWizardDisp = dispinterface;
  FormatWizard = IFormatWizard;
```

```
  IFormatWizard = interface(IDispatch)
    ['{0AAEE702-DAFB-11D7-8CF1-992A98189A54}']
    function  Execute(const DwgFile: WideString;
                      const ResIniFile: WideString;
                      const DefDir: WideString;
                      const TemplDir: WideString): OleVariant; safecall;
  end;

IFormatWizardDisp = dispinterface
    ['{0AAEE702-DAFB-11D7-8CF1-992A98189A54}']
    function  Execute(const DwgFile: WideString;
                      const ResIniFile: WideString;
                      const DefDir: WideString;
                      const TemplDir: WideString): OleVariant; dispid 1;
  end;

  CoFormatWizard = class
    class function Create: IFormatWizard;
    class function CreateRemote(const MachineName: string): IFormatWizard;
  end;

implementation

uses ComObj;

class function CoFormatWizard.Create: IFormatWizard;
begin
  Result := CreateComObject(CLASS_FormatWizard) as IFormatWizard;
end;

class function CoFormatWizard.CreateRemote(const MachineName: string):
IFormatWizard;
begin
  Result := CreateRemoteComObject(MachineName, CLASS_FormatWizard) as
IFormatWizard;
end;

end.
```

Реализация метода записана нами в файл u_FomatWizard.pas (листинг 21.12), основная часть которого сгенерирована Delphi.

Листинг 21.12. Файл u_FomatWizard.pas

```
unit u_FomatWizard;

interface

uses
  ComObj, ActiveX, ruFormatWizardSrv_TLB, StdVcl;

type
  TFormatWizard = class(TAutoObject, IFormatWizard)
```

```
  protected
    function Execute(const DwgFile, ResIniFile, DefDir,
          TemplDir: WideString): OleVariant; safecall;
    { Protected declarations }
  end;

implementation

uses ComServ, frmFormatWizard, Forms, Controls, Dialogs;

function TFormatWizard.Execute(const DwgFile, ResIniFile, DefDir,
                               TemplDir: WideString): OleVariant;
var
  Wizard: TFormWizard;
begin
  Wizard := TFormWizard.Create(Application);
  with Wizard do
  begin
    DwgFileName := DwgFile;
    ResultIniFile := ResIniFile;
    FormatDefDir := DefDir;
    TemplateDir := TemplDir;
    Result := (ShowModal = mrOk);
    Free;
  end;

end;

initialization
  TAutoObjectFactory.Create(ComServer, TFormatWizard, Class_FormatWizard,
                                        ciMultiInstance, tmApartment);
end.
```

Вот и все, что надо было сделать на Delphi.

Разработка LISP-программы

Программу вычерчивания формата мы будем писать на LISP. Но сначала нам нужно создать набор блоков основных надписей. Перечень необходимых блоков мы составили в начале главы.

Создание блоков основных надписей

Блоки основных надписей нужно делать очень аккуратно, в полном соответствии с ГОСТ. Единицей рисунка при создании этих блоков является миллиметр. Линии блоков рисуем отрезками, назначая нулевой вес для тонких линий разделяющих строки и вес 0,50 мм для основных линий. Особое внимание следует уделять атрибутам. Имена атрибутов должны точно соответствовать программе. Размещение атрибутов необходимо выполнять не "на глаз", а с использованием объектной привязки.

Лучше всего сначала нарисовать блок основной надписи для чертежей зданий, а потом, копируя его и редактируя, создать остальные блоки. Точка вставки блоков основных надписей должна быть в правом нижнем углу внутренней рамки формата, а блоков инвентарного номера и согласований — в левом нижнем углу внутренней рамки формата. Пример блока основной надписи для чертежей зданий и сооружений ru_format_base.dwg представлен на рис. 21.11. На рисунке видны имена и расположение атрибутов.

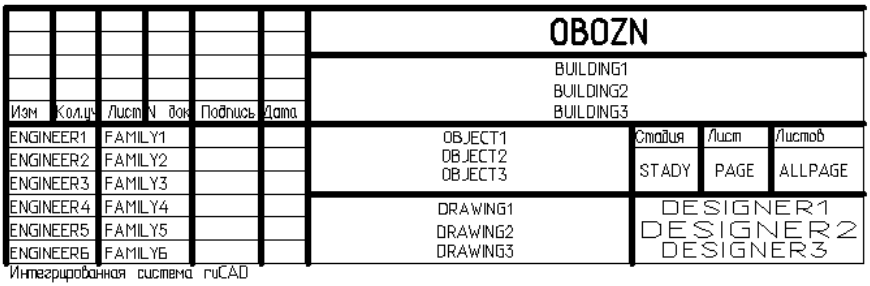


Рис. 21.11. Блок основной надписи чертежей зданий и сооружений

Примечание

Иногда обнаруживается недостаток оформления атрибутов блока. Например, для длинных фамилий желательно использовать в качестве выравнивания текста не **Left** (Левый), а **Fit** (По ширине). В этом случае необходимо модернизировать фрагмент программы, отвечающий за создание основной надписи, с заменой некоторых атрибутов блока на текстовые примитивы. *(Николай Полещук)*

Программа рисования формата

Переходим к заключительному этапу — программе рисования формата. Эту программу мы записываем в файл ru_draw_format.lsp (листинг 21.13).

Листинг 21.13. Файл ru_draw_format.lsp

```
(defun START (/ base_frm blocks_section block_ent block_ins_pnt
            format_block_dir format_default_ini format_dir format_height
        format_logo_dir format_section format_width form_type ins_scale
      in_left_bottom_pnt in_left_top_pnt in_rect_ent in_right_bottom_pnt
 in_right_top_pnt izd_frm left_block_ins_pnt list_attr_values logo_block
              next_frm out_left_bottom_pnt out_left_top_pnt out_rect_ent
         out_right_bottom_pnt out_right_top_pnt pnt1 pnt2 pnt3 reg_block
 result_file res_srv selection_frm selection_out sogl_block sogl_reguired
 srv stamp_block text_frm)
;;; Служебная функция чтения из INI-файла списка переменных
  (defun _list-regular-var (ini_file section name_var steps / i result)
;| Возвращает список пар "переменная-значение". Пример:
(_list-regular-var "testFormat.frm" "Format" "Building" 3)
```

```
Вернет
(("Building1" . "Бывшая гостиница \"Каир\"")
  ("Building2" . "по ул. Воронья Слободка")
  ("Building3" . "в г. Черноморске")
)
|;
    (setq i 1 result '())
    (repeat steps
      (setq result (append result
                           (list
                             (cons
                               (strcat name_var (itoa i))
                               (ru-ini-read
                                 ini_file
                                 section
                                 (strcat name_var (itoa i))
                                 ""
                               );_ end of ru-ini-read
                             );_ end of list
                           );_ end of list
                 );_ end of append
            i      (1+ i)
     );_ end of setq
   );_ end of repeat
    result
);_ end of defun
;;; ####### Главная функция #######
 (ru-app-begin)
 (if (setq srv (vlax-get-or-create-object
                                  ruFormatWizardSrv.FormatWizard"))
  (progn
      (setq format_dir (strcat (ru-dirs-get-local-app-data) "Format\\")
            format_default_ini (strcat format_dir "DefaultFormat.ini")
            format_block_dir   (strcat format_dir "Blocks\\")
            format_logo_dir    (strcat format_dir "Logo\\")
            blocks_section     "Blocks"
            base_frm           "0"
            izd_frm            "1"
            text_frm           "2"
            next_frm           "3"
            format_section     "Format"
            result_file        (ru-file-dwgname-type ".frm")
;;; Вызываем Мастер формата
            res_srv            (vlax-invoke-method
                                 srv
                                 "Execute"
                                 (ru-file-full-dwgname)
                                 result_file
                                 (strcat (ru-dirs-get-local-app-data)
                                                    "Format\\")
```

```
                                (ru-file-template "")
                              );_ end of vlax-invoke
     );_ end of setq
;;; Анализируем результат. Если выход из Мастера был по кнопке
;;; Чертить, результат будет :vlax-true
      (if (= (vlax-variant-value res_srv) :vlax-true)
        (progn
;;; Читаем результаты из файла
;;; Ширина формата
          (setq format_width (atof
            (ru-ini-read result_file format_section "Width" "0"))
;;; Высота формата
                format_height (atof (ru-ini-read result_file
                                      format_section "Height" "0")))
;;; Дополнительная проверка размеров формата
          (if (and (> format_width 0) (> format_height 0))
            (progn
;;; Тип блока основной надписи
              (setq form_type (ru-ini-read result_file format_section
                                "FormType" base_frm)
                    stamp_block (ru-ini-read format_default_ini
                        blocks_section (strcat "StampBlock" form_type)
                                "")
                    reg_block (ru-ini-read format_default_ini
                        blocks_section "RegisterBlock" ""))
;;; Требуется ли блок согласований
              (if (setq sogl_reguired (ru-conv-str-to-bool
                          (ru-ini-read result_file format_section
                          "SoglReguired" "0")))
                (setq sogl_block (ru-ini-read format_default_ini
                         blocks_section "SoglBlock" ""))
              );_ end of if
;;; Рисование
;;; Наружная рамка. Сначала рисуем формат с левым нижним углом в точке;;; 0,0
              (setq out_left_bottom_pnt
                     '(0.0 0.0)
                    out_right_bottom_pnt (polar out_left_bottom_pnt 0
                            (ru-conv-millimeter-in-paper-to-unit
                            format_width))
                    out_left_top_pnt (polar out_left_bottom_pnt
                            (ru-geom-go-left 0)
                            (ru-conv-millimeter-in-paper-to-unit
                             format_height))
                    out_right_top_pnt (polar out_left_top_pnt 0
                            (ru-conv-millimeter-in-paper-to-unit
                            format_width))
              );_ end of setq
;;; Рисуем наружную рамку
              (if (setq out_rect_ent
                        (ru-pline-add
```

```
                          (list out_left_bottom_pnt
                                out_right_bottom_pnt
                                out_right_top_pnt
                                out_left_top_pnt
                          );_ end of list
                          t 0 0 "CONTINUOUS"))
                (progn
;;; Рисуем рамочку для номеров страниц
;;; При этом создаем набор из внешней рамки, для того чтобы
;;; все, что будет нарисовано после него, можно было бы добавить
;;; в общий набор примитивов блока
                 (setq selection_out (ssadd out_rect_ent (ssadd))
                       pnt1 (polar out_right_top_pnt (ru-geom-go-back 0)
                        (ru-conv-millimeter-in-paper-to-unit 10.0))
                       pnt2 (polar pnt1 (ru-geom-go-right 0)
                            (ru-conv-millimeter-in-paper-to-unit 5.0))
                      pnt3 (polar out_right_top_pnt (ru-geom-go-right 0)
                             (ru-conv-millimeter-in-paper-to-unit 5.0)))
               (ru-pline-add (list pnt1 pnt2 pnt3) nil 0 0 "CONTINUOUS")
;;; Внутренняя рамка
                (setq in_left_bottom_pnt
                       (list (ru-conv-millimeter-in-paper-to-unit 20.0)
                             (ru-conv-millimeter-in-paper-to-unit 5.0)
                       );_ end of list
                       in_right_bottom_pnt
                           (polar  in_left_bottom_pnt 0
                           (ru-conv-millimeter-in-paper-to-unit
                            (- format_width 25.0)))
                       in_left_top_pnt
                        (polar in_left_bottom_pnt (ru-geom-go-left 0)
                               (ru-conv-millimeter-in-paper-to-unit
                                 (- format_height 10.0)))
                       in_right_top_pnt
                        (polar in_left_top_pnt 0
                          (ru-conv-millimeter-in-paper-to-unit
                           (- format_width 25.0)))
;;; Это будет точка вставки блоков основной надписи
                       block_ins_pnt in_right_bottom_pnt
;;; Это будет точка вставки блоков инвентарных номеров и согласований
                       left_block_ins_pnt in_left_bottom_pnt
                 );_ end of setq
;;; Рисуем внутреннюю рамку полилинией с "нормальным" весом
                 (if (setq in_rect_ent
                           (ru-pline-add
                             (list in_left_bottom_pnt
                                   in_right_bottom_pnt
                                   in_right_top_pnt
                                   in_left_top_pnt
                            );_ end of list
                            t 0 (ru-lw-normal) "CONTINUOUS"))
```

```
                    (progn
;;; Переходим к вставке блоков
;;; Определяем масштаб вставки с учетом, что формат может рисоваться
;;; и в пространстве листа, и в пространстве модели
;;; Сдвигаем форматированный текст влево, иначе займем в книге
;;; слишком много места
  (setq ins_scale  (ru-conv-millimeter-in-paper-to-unit 1.0))
  (if (and stamp_block (/= stamp_block ""))
    (progn
      (setq block_ent
         (ru-block-insert-obj (ru-file-set-ext (strcat format_block_dir
                                stamp_block) ".dwg")
            block_ins_pnt ins_scale ins_scale ins_scale 0))
;;; Установка значений атрибутов
;;; Проверяем, задан ли блок логотипа
      (setq logo_block (ru-ini-read result_file format_section
                "LogoBlock" "")
;;; Начинаем формировать список значений всех атрибутов блока
;;; По каждому атрибуту создаем список вида ("ИМЯ" . "ЗНАЧЕНИЕ")
;;; и соединяем их в общий список list_attr_values
          list_attr_values
;;; Обозначение документа
            (append
              (list (cons "OBOZN" (ru-ini-read result_file
                                          format_section "Obozn" "")))
;;; Лист
              (list (cons "PAGE" (ru-ini-read result_file
                                          format_section "Page" "")))
;;; Листов
              (list (cons "ALLPAGE" (ru-ini-read result_file
                                       format_section "PageTotal" "")))
;;; Стадия
              (list (cons "STADY" (ru-ini-read result_file
format_section "Stady" "")))
           );_ end of append
        );_ end of setq
;;; Далее добавление атрибутов зависит от формы основной надписи
        (if (= form_type izd_frm)
            (setq list_attr_values
;;; Масштаб, масса, наименование и материал изделия проставляются
;;; только у чертежей изделий
            (append list_attr_values
               (list (cons "SCALE" (ru-ini-read result_file
                                format_section "Scale" "")))
               (list (cons "MASSA" (ru-ini-read result_file
                               format_section "Massa" "")))
;;; Пять строк наименования изделия
              (_list-regular-var result_file format_section "Izd" 5)
;;; Три строки материала изделия
              (_list-regular-var result_file format_section "Material" 3)
          );_ end of append
```

```
        );_ end of setq
      );_ end of if
;;; Обычная основная надпись
      (if (= form_type base_frm)
        (setq list_attr_values (append list_attr_values
;;; Три строки наименования стройки
          (_list-regular-var result_file format_section "Building" 3)
;;; Три строки наименования объекта
          (_list-regular-var result_file format_section "Object" 3)
                             )))
;;; Три строки наименования чертежа
      (if (/= form_type izd_frm)
        (setq list_attr_values (append list_attr_values
          (_list-regular-var result_file format_section "Drawing" 3)))
     );_ end of if
;;; Если это не чертеж продолжения текста
      (if (/= form_type next_frm)
        (setq list_attr_values (append list_attr_values
;;; Добавляем шесть строк подписей
         (_list-regular-var result_file format_section "Engineer" 6)
         (_list-regular-var result_file format_section "Family" 6)
;;; Если блок логотипа не задан, заполняем текстовое наименование
;;; проектной организации
         (if (= logo_block "")
          (_list-regular-var result_file format_section "Designer" 3))
         )))
;;; А теперь разом изменяем значение всех атрибутов на требуемые
         (ru-block-change-attributes block_ent list_attr_values)
;;; Если блок логотипа был задан, вставляем его
         (if (/= logo_block "")
           (ru-block-insert-obj (ru-file-set-ext
                (strcat format_logo_dir logo_block) ".dwg")
                  block_ins_pnt ins_scale ins_scale ins_scale 0))
      );_ end of progn
     );_ end of if
;;; Вставка блока инвентарного номера
      (if (and reg_block (/= reg_block ""))
        (ru-block-insert-obj (ru-file-set-ext
             (strcat format_block_dir reg_block) ".dwg")
            left_block_ins_pnt ins_scale ins_scale ins_scale 0))
;;; Вставка блока согласований
      (if (and sogl_block (/= sogl_block ""))
        (ru-block-insert-obj (ru-file-set-ext
             (strcat format_block_dir sogl_block) ".dwg")
            left_block_ins_pnt ins_scale ins_scale ins_scale 0))
;;; Нарисовано все, что только можно
;;; Все нарисованное объединяем в один набор
    (setq selection_frm (ru-ss-join selection_out
                              (ru-ss-select-after-ent out_rect_ent)))
;;; Если пользователь хочет передвинуть формат, позволяем это
   (if (ru-dlg-yes-cml "Передвинуть формат")
```

```
    (progn
      (princ (strcat "\n Левый нижний угол формата  <"
              (rtos (car out_left_bottom_pnt)) ","
              (rtos (cadr out_left_bottom_pnt)) ">: "))
;;; В данном случае пользуемся функцией command для визуального
;;; отслеживания перемещения формата
      (command "_.MOVE" selection_frm "" out_left_bottom_pnt pause)
   );_ end of progn
  );_ end of if
;;; Если пользователь хочет повернуть формат, позволяем это
   (if (not (ru-dlg-no-cml "Повернуть формат"))
    (progn
      (princ "\nУгол поворота: ")
      (command "_.ROTATE" selection_frm "" (getvar "LASTPOINT") pause)
   );_ end of progn
  );_ end of if
;;; Для экономии места завершаем все скобки в одной строке
)))))
 (alert
  (strcat "\nНеверная ширина или высота: Ширина=" (rtos format_width)
                           " Высота=" (rtos format_height)))
))))
 (alert "\nНе удалось создать ruFormatWizardSrv.FormatWizard")
)
(ru-app-end)
(princ)
);_ end of defun
(START)
```

Текст функции изменения значения атрибутов блока представлен в листинге 21.14.

Листинг 21.14. Функция `ru-block-change-attributes`

```
(defun ru-block-change-attributes (block_ent att_list / blk itm atts)
;|
Изменение значений атрибутов блока. Параметры:
block_ent - примитив или vla-object блока
att_list - список точечных пар "имя_атрибута" . "значение_атрибута"
|;
 (if (= (type block_ent) 'vla-object)
    (setq blk block_ent)
    (setq blk (vlax-ename->vla-object block_ent)
   );_ end of setq
);_ end of if
  (if (= (vla-get-hasattributes blk) :vlax-true)
    (progn
      (setq atts
            (vlax-safearray->list
              (vlax-variant-value (vla-getattributes blk))
            );_ end of vlax-SafeArray->list
     );_ end of setq
```

```
    (foreach item att_list
       (mapcar
         '(lambda (x)
            (if
              (= (strcase (car item)) (strcase (vla-get-tagstring x)))
               (vla-put-textstring x (cdr item))
           );_ end of if
         );_ end of lambda
         atts
       );_ end of mapcar
     );_ end of foreach
      (vla-update blk)
   );_ end of progn
 );_ end of if
)
```

Все! Загружаем файл ru_draw_format.lsp в AutoCAD. При этом автоматически выполняется функция START, появляется диалоговое окно Мастера, мы делаем несколько щелчков при установке опций, на финишной странице щелкаем кнопку **Чертить**, в рисунке появляется нарисованный формат, при необходимости передвигаем и поворачиваем его и любуемся результатом работы (рис. 21.12).

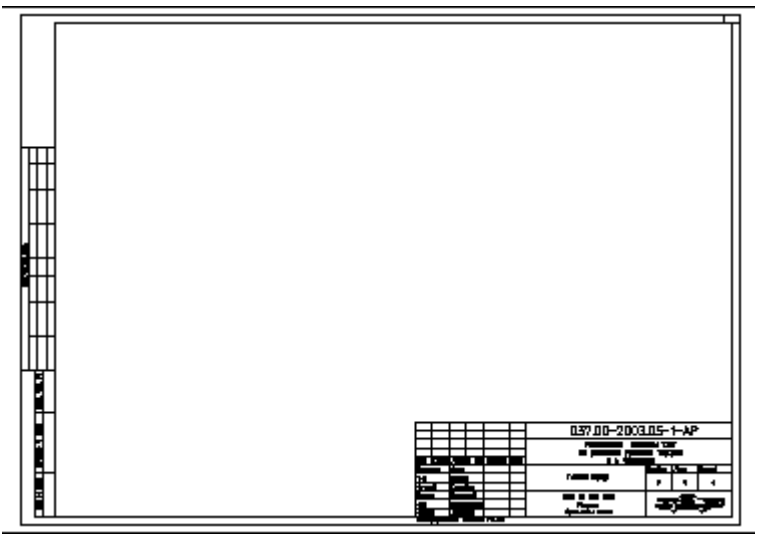


Рис. 21.12. Нарисованный формат

Увеличив рисунок, рассматриваем, как выглядит основная надпись при отключенном (рис. 21.13) и включенном (рис. 21.14) режиме **LWT** (ВЕС).

Логотип фирмы, конечно, корявый и непонятный, но за него пусть отвечает директор фирмы господин Фунт — ему не привыкать. Более внимательно рассмотрев рисунок, видим ошибку — в графе **Лист** стоит число 4, а в графе **Листов** — 1. Разыскиваем ошибку в программе. Просмотрев исходные тексты и файл результата рабо-

ты Мастера, обнаруживаем, что ошибка спряталась на уровне формы Мастера. Если номер листа не первый, то количество листов должно возвращаться пустым, хотя в редакторе имеется какое-то число. Вносим исправления в код (листинг 21.15), перекомпилируем проект в Delphi и убеждаемся, что теперь все правильно.

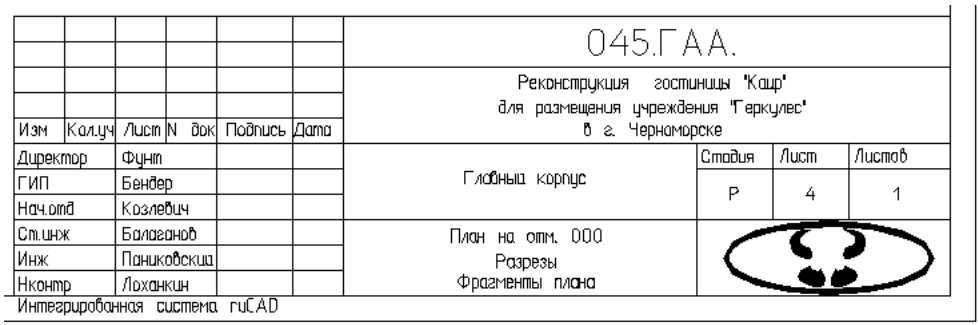


Рис. 21.13. Основная надпись при отключенном режиме **LWT**

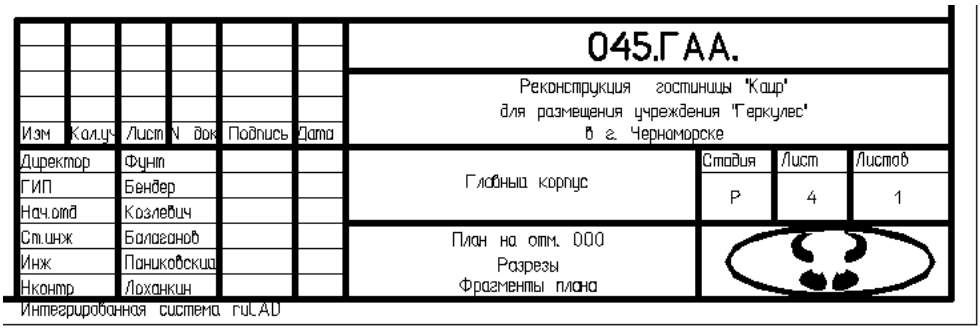


Рис. 21.14. Основная надпись при включенном режиме **LWT**

Листинг 21.15. Устранение ошибки количества листов. Фрагмент процедуры `MakeResult`

```
procedure TFormWizard.MakeResult;
var
  BoolS: string;
begin
......
SaveStringToIni(ResultIniFile, FormatSectionName,
    FormatListVarName, SpinEditList.Text);
if SpinEditList.Text='1' then BoolS := ''
                  else BoolS := SpinEditListAll.Text;
  SaveStringToIni(ResultIniFile, FormatSectionName,
    FormatListAllVarName, BoolS);
end;
```

Включение программы в меню

Для полного счастья осталось записать готовую программу в меню. Включаем ее в XML-меню (листинг 21.16).

Листинг 21.16. Фрагмент файла Tables.ruxm

```
<?xml version='1.0' encoding='windows-1251'?>
<root name='Таблицы' macro=''>
  <item name='Формат'
        image='draw\format.gif'
        comment='Мастер рисования формата'
        macro='(ru-app-load "ru_draw_format")'/>
<item name='Спецтаблицы' macro=''>
 <include='tabl_gp.ruxm'  macro='Таблицы ГП'/>
 <include='tabl_ar.ruxm'  macro='Таблицы АР'/>
 <include='tabl_ov.ruxm'  macro='Таблицы ОВ'/>
 <include='tabl_wk.ruxm'  macro='Таблицы ВК'/>
 <include='tabl_tx.ruxm'  macro='Таблицы ТХ'/>
 <include='tabl_gs.ruxm'  macro='Таблицы ГАЗ'/>
 <include='tabl_eo.ruxm'  macro='Таблицы ЭЛЕКТРО'/>
 <include='tabl_to.ruxm'  macro='Таблицы ТОПО'/>
</item>
</root>
```

ГЛАВА 22

Завершение разработки главной библиотеки

После разработки дополнительных библиотек с использованием COM-технологий мы могли бы приступить к разработке прикладных программ. Однако нам потребуется еще много универсальных функций, позволяющих сделать программы проще. Кроме того, много функций требуется для поддержки общей идеологии системы, продуманной в начале книги. Так что спешить с программами мы не будем, а дополним свою библиотеку функций. Библиотеку функций невозможно написать "раз и навсегда", она будет непрерывно корректироваться и пополняться, но основной набор мы должны сформировать заранее.

Конечно, все функции мы не будем разбирать из-за ограниченного объема издания. Остановимся только на самых важных и интересных — остальные можно найти на прилагаемом компакт-диске.

Ширина и вес линий

В *главе 3* мы детально продумали стратегию работы с шириной и весом линий с учетом достоинств и недостатков различных способов отображения ширины, и в качестве основного способа управления отображением было намечено изменение веса линий. Теперь нам предстоит реализовать задуманное. И физическую ширину полилиний (когда это требуется), и вес линий пользователю удобнее задавать в миллиметрах на бумаге. Такая возможность предоставляется AutoCAD при выборе веса линий. Внутри AutoCAD текущий вес линий содержится в системной переменной CELWEIGHT, а вес по умолчанию — в системной переменной LWDEFAULT. Управление весом линий, с одной стороны, упрощается тем, что пользователь всегда может изменить текущий вес с помощью весьма наглядного раскрывающегося списка **Lineweight** (Веса) панели **Object Properties** (Свойства объектов)[1], с другой стороны, осложняется тем, что неразумный пользователь может применять это мощное средство необдуманно.

В наших программах вес линий, там где это важно, будет задаваться достаточно жестко, но пользователь будет иметь возможность изменить вес путем выбора из огра-

[1] В системах AutoCAD 2004 и 2005 — панели **Properties** (Свойства).

ниченного списка весов, соответствующих нашему стандарту. В предлагаемом списке для ширины будут присутствовать (все приведенное к "мм на бумаге"):

- "Текущая в ruCAD: 0.25 мм" — действующий вес;
- "Тонкая: 0.00 мм" — вес, соответствующий тонкой линии;
- "Утолщенная: 0.25 мм" — вес, соответствующий утолщенной линии (дополнение, внесенное "по просьбам трудящихся");
- "Основная: 0.50 мм" — вес, соответствующий основной линии (ширина основной линии по умолчанию может быть изменена в файле настройки, например на 0.8 мм на бумаге, если кому-то так больше нравится);
- "Как в AutoCAD: 0.03 мм (по слою)" — текущая в AutoCAD.

Пользователь может установить вес и "как в AutoCAD", но сделает это уже более осознанно.

При выполнении всех "рисовальных" функций мы будем в аргументах передавать требуемый вес линий. Для частей изображений, рисуемых "всегда тонко", вес будет задаваться равным нулю, а для рисуемых и "тонко", и "толсто" — в виде значения, возвращаемого функцией `ru-lw-current` (листинг 22.1).

Листинг 22.1. Функция `ru-lw-current`

```
(defun ru-lw-current ()
;|-------------------------------------------------------------------------
Возвращает текущий вес линии, предлагаемый по умолчанию. Отличается
от (getvar "CELWEIGHT") тем, что может изменяться только из нашего диалога.
 Возвращает вес в единицах LW!!!
Пример: (ru-lw-current) > 60
-------------------------------------------------------------------------|;
 (cond
  (*ru_current_lw*)
  (T
   (setq *ru_current_lw*
    (ru-conv-millimeter-to-acadlw
     (atof (ru-ini-read-default nil "Setup" "CurrentLineWeight" "0.5")))
  );_ end of setq
 )
);_ end of cond
);_ end of defun
```

Для преобразования ширины из "мм на бумаге" во внутренний формат веса в системе AutoCAD потребуется специальная функция (листинг 22.2).

Листинг 22.2. Функция `ru-conv-millimeter-to-acadlw`

```
(defun ru-conv-millimeter-to-acadlw (mm / tmp)
;|-------------------------------------------------------------------------
Перевод ширины в миллиметрах в единицы LW AutoCAD
Примеры:
(ru-conv-millimeter-to-acadlw 0.0) > 0
(ru-conv-millimeter-to-acadlw 0.1) > 13
```

```
(ru-conv-millimeter-to-acadlw 0.5) > 50
(ru-conv-millimeter-to-acadlw 5) > 211
------------------------------------------------------------------------|;
  (setq tmp (ru-match-round (* mm 100)))
  (cond
    ((< tmp 0) 0)
    ((> tmp 211) 211)
    (T
    ;; округлить до ближайшего значения из списка
     (ru-match-nearest-from-list tmp (ru-lw-std-lst) "AUTO")
   )
 );_ end of cond
);_ end of defun
```

Тексты еще нескольких функций, оперирующих весами, приведены в листингах 22.3—22.8.

Листинг 22.3. Функция `ru-lw-std-lst`

```
(defun ru-lw-std-lst ()
;|------------------------------------------------------------------------
Список стандартных весов линий в виде именованных констант AutoCAD
------------------------------------------------------------------------|;
 (list
 aclnwt000 aclnwt005 aclnwt009 aclnwt013 aclnwt015 aclnwt018 aclnwt020
 aclnwt025 aclnwt030 aclnwt035 aclnwt040 aclnwt050 aclnwt053 aclnwt060
 aclnwt070 aclnwt080 aclnwt090 aclnwt100 aclnwt106 aclnwt120 aclnwt140
 aclnwt158 aclnwt200 aclnwt211)
);_ end of defun
```

Листинг 22.4. Функция `ru-lw-conv-celweight-to-mm`

```
(defun ru-lw-conv-celweight-to-mm (/ lweight)
;|------------------------------------------------------------------------
 Конвертирование веса линий в миллиметры
------------------------------------------------------------------------|;
  (setq lweight (getvar "CELWEIGHT"))
  (cond
    ((or (= lweight -1) (= lweight -2));по слою или по блоку
    ;;Выясняем вес для слоя
     (abs (* 0.01 (ru-layer-lweight-current-layer)))
   )
    ((= lweight -3)          ; по умолчанию
     (* 0.01 (getvar "LWDEFAULT"))
   )
    (T (* 0.01 lweight))
 );_ end of cond
);_ end of defun
```

Листинг 22.5. Функция `ru-lw-conv-celweight-to-string`

```
(defun ru-lw-conv-celweight-to-string (/ lweight)
;|-------------------------------------------------------------------------
Конвертирование веса линий в строки для отображения в диалоге
-------------------------------------------------------------------------|;
  (setq lweight (getvar "CELWEIGHT"))
  (cond
    ((= lweight -1)          ; по слою
     (strcat (rtos (* 0.01 (abs (ru-layer-lweight-current-layer))) 2 2)
             " мм (по слою)"
    );_ end of strcat
   )
    ((= lweight -2)          ; по блоку
     (strcat (rtos (* 0.01 (abs (ru-layer-lweight-current-layer))) 2 2)
             " мм (по блоку)"
    );_ end of strcat
   )
    ((= lweight -3)          ; по умолчанию
     (strcat (rtos (* 0.01 (abs (getvar "LWDEFAULT"))) 2 2)
             " мм (по умолчанию)"
    );_ end of strcat
   )
    (T (strcat (rtos (* 0.01 lweight) 2 2) " мм"))
 );_ end of cond
);_ end of defun
```

Листинг 22.6. Функция `ru-lw-make-double-lst`

```
(defun ru-lw-make-double-lst ()
;|-------------------------------------------------------------------------
Создание текущего двойного списка. В первом подсписке - названия для выбора,
во втором - соответствующие значения веса. Предназначены для диалога выбора
Пример:
(ru-lw-make-double-lst)
(("Текущая в ruCAD:              0.25 мм"
   "Тонкая:                      0.00 мм"
   "Утолщенная:                  0.25 мм"
   "Основная:                    0.5 мм"
   "Как в AutoCAD:               0.03 мм (по слою)"
) (25 0 25 50 -1))
-------------------------------------------------------------------------|;
  (list
    (list
      (strcat (ru-sring-align-to-left "Текущая в ruCAD:" 30)
              (rtos (* 0.01 (ru-lw-current)) 2 2) " мм")
      (strcat (ru-sring-align-to-left "Тонкая:" 38) "0.00 мм")
      (strcat (ru-sring-align-to-left "Утолщенная:" 33)
              (rtos (* 0.005 (ru-lw-normal)) 2 2) " мм")
```

```
    ;; половина основной
     (strcat (ru-sring-align-to-left "Основная:" 36)
             (rtos (* 0.01 (ru-lw-normal)) 2 2) " мм")
    ;;  - заданная по умолчанию (обычно 0.5 мм)
     (strcat (ru-sring-align-to-left "Как в AutoCAD:" 33)
             (ru-lw-conv-celweight-to-string))
  );_ end of list
   (list (ru-lw-current)
         0
         (ru-conv-millimeter-to-acadlw (* 0.005 (ru-lw-normal)))
         (ru-lw-normal)
         (getvar "CELWEIGHT")
  );_ end of list
 );_ end of list
);_ end of defun
```

Листинг 22.7. Функция `ru-lw-set-for-ent`

```
(defun ru-lw-set-for-ent (ent lweight)
;|-------------------------------------------------------------------------
 Установка веса линии для примитива
-------------------------------------------------------------------------|;
  (ru-ent-mod ent lweight 370)
);_ end of defun
```

Листинг 22.8. Функция `ru-lw-set-for-ss`

```
(defun ru-lw-set-for-ss (ss lweight)
;|-------------------------------------------------------------------------
Установка веса линии для набора
-------------------------------------------------------------------------|;
  (ru-ss-mod ss lweight 370)
);_ end of defun
```

Теперь мы можем разработать функцию с диалоговым окном для установки веса линий (рис. 22.1, листинги 22.9 и 22.10).

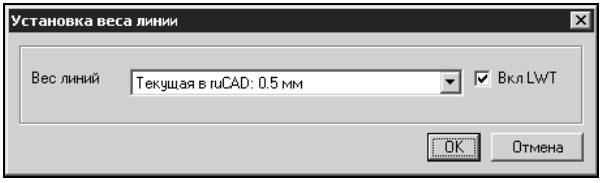


Рис. 22.1. Диалоговое окно установки веса линии

Листинг 22.9. Функция `ru-dlg-dcl-select-lw`

```
(defun ru-dlg-dcl-select-lw (/ index_dlg lw_double_lst lw_lst lw_popup_lst
popup_list_lineweight result toggle_lwt what_next
 _select-lw-what-next _select-lw-save-params _select-lw-action-tile
```

```
_select-lw-popup-list-lineweight _select-lw-toggle-lwt
_select-lw-set-tile _select-lw-init-params)
;;; -------- Локальная функция -------------
  (defun _select-lw-init-params ()
    (setq lw_double_lst (ru-lw-make-double-lst)
          lw_popup_lst  (car lw_double_lst)
          lw_lst        (cadr lw_double_lst)
   );_ end of setq
 );_ end of defun
;;; -------- Локальная функция -------------
  (defun _select-lw-set-tile ()
    (start_list "popup_list_lineweight")
    (mapcar 'add_list lw_popup_lst)
    (end_list)
   ;; Установить позицию по текущему значению
    (set_tile "popup_list_lineweight" "0")
    (setq toggle_lwt (itoa (getvar "LWDISPLAY")))
    (set_tile "toggle_lwt" toggle_lwt)
 );_ end of defun
;;; -------- Локальная функция -------------
  (defun _select-lw-toggle-lwt (value)
    (setq toggle_lwt value)
 );_ end of defun
;;; -------- Локальная функция -------------
  (defun _select-lw-popup-list-lineweight (value)
    (setq popup_list_lineweight value)
 );_ end of defun
;;; -------- Локальная функция -------------
  (defun _select-lw-action-tile ()
    (action_tile "popup_list_lineweight"
     "(_select-lw-popup-list-lineweight $value)")
    (action_tile "toggle_lwt" "(_select-lw-toggle-lwt $value)")
    (action_tile "accept" "(done_dialog 1)")
    (action_tile "cancel" "(done_dialog 0)")
 );_ end of defun
;;; -------- Локальная функция -------------
  (defun _select-lw-save-params ()
    (ru-lw-save-current-to-dwg)
 );_ end of defun
;;; -------- Локальная функция -------------
  (defun _select-lw-what-next ()
    (cond
      ((= 0 what_next)
       nil
     )
      ((= 1 what_next)
       (cond
        ;; 0 пропускаем, ничего не изменилось
         ((= popup_list_lineweight "1")
          (setq *ru_current_lw* 0)
        )
```

```
         ((= popup_list_lineweight "2")
         ;; Половина нормы
          (setq *ru_current_lw*
;| Если норму разделим пополам, могут получиться дробные значения, поэтому норму
в единицах LW превращаем в мм и конвертируем обратно в LW
|;
                (ru-conv-millimeter-to-acadlw
                  (* 0.005 (ru-lw-normal))
                );_ end of ru-conv-millimeter-to-acadlw
         );_ end of setq
        )
         ((= popup_list_lineweight "3")
         ;; Устанавливаем норму
          (setq *ru_current_lw* (ru-lw-normal))
        )
         ((= popup_list_lineweight "4")
;| Как в AutoCAD. Конвертировать в мм CELWEIGHT и обратно.
Нам не нужны -1, -2 и -3, необходима конкретная величина|;
          (setq *ru_current_lw*
                (ru-conv-millimeter-to-acadlw
                  (ru-lw-conv-celweight-to-mm)
                );_ end of ru-conv-millimeter-to-acadlw
         );_ end of setq
        )
      );_ end of cond
       (setvar "LWDISPLAY" (atoi toggle_lwt))
       (_select-lw-save-params)
       *ru_current_lw*
     )
   );_ end of cond
 );_ end of defun
 ;; ------------ Главная функция --------------------
  (_select-lw-init-params)
  (setq what_next 1000)
  (if (> (setq index_dlg (load_dialog (ru-file-dcl "ru_get_lw"))) 0)
    (progn
      (while (> what_next 1)
       ;; вывод диалога на экран пока не нажата OK или CANCEL
        (if (new_dialog "get_lineweight" index_dlg)
          (progn
            (_select-lw-set-tile)
            (_select-lw-action-tile)
           ;; Запуск в цикл ожидания
            (setq what_next (start_dialog)
                  result    (_select-lw-what-next)
            );_ end of setq
         );_ end of progn
          (progn
            (setq what_next 0)
            (ru-msg-alert "Не могу показать диалоговое окно!")
         );_ end of progn
```

```
       );_ end of if
     );_ end of while
      (unload_dialog index_dlg)
   );_ end of progn
    (alert
      (strcat "Не могу загрузить " (ru-file-dcl "ru_get_lw"))
   )
 );_ end of if
  result
);_ end of defun
```

Листинг 22.10. Файл ru_get_lw.dcl

```
dcl_settings : default_dcl_settings { audit_level = 0; }
@include "ru_base.dcl"
get_lineweight : dialog {
    label = "Установка веса линии";
    key = "title";
    :boxed_column {
        :row {
        : list_box {
                        label = "Вес линий";
                        key = "popup_list_lineweight";
                        fixed_width = true;
                        fixed_height = true;
                        height = 7;
                       width = 40;
        }
                : toggle {
                label = "Вкл LWT";
                key = "toggle_lwt";
                value = "0";
                }
        }
 errtile;
}
   _ok_cancel;
}
```

Вот теперь мы можем в любой подходящий момент вызвать функцию `ru-dlg-dcl-select-lw` и установить текущий вес до следующего изменения.

Функции для геометрических построений

В программах "рисовального" типа нам придется тысячи раз вычислять координаты точек. Никаких сложностей в этом нет, но для предотвращения механических ошибок мы инкапсулируем повторяющиеся вычисления в функции. Для примера приведем несколько функций такого класса (листинги 22.11—22.13).

Листинг 22.11. Функция ru-geom-go-left

```
(defun ru-geom-go-left (ang)
;;; Возвращает угол влево на 90 градусов от заданного
  (+ ang (/ pi 2))
)
```

Листинг 22.12. Функция ru-geom-go-right

```
(defun ru-geom-go-right (ang)
;;; Возвращает угол вправо на 90 градусов от заданного
  (- ang (/ pi 2))
)
```

Листинг 22.13. Функция ru-geom-go-back

```
(defun ru-geom-go-back (ang)
;;; Возвращает угол в противоположном направлении от заданного
  (+ ang pi)
)
```

Разумеется, нужны и более серьезные функции для работы с геометрией объектов, но мы будем публиковать их при необходимости.

Учет особенностей систем координат

Прежде чем перейти к функциям рисования, нам нужно разобраться с системами координат. *Все функции, использующие объектную модель, работают только с мировой системой координат (МСК, WCS).*

Если забыть про это обстоятельство, можно наделать много трудно обнаруживаемых ошибок. Обычно при разработке программ авторы используют только МСК и у них все работает правильно, и лишь "неправильные" пользователи потом удивляются, почему объекты появляются не там, где нужно. При использовании функции `command` все рисование выполняется в активной *пользовательской системе координат* (ПСК, UCS) и заботиться о пересчете координат не нужно. Впредь мы постараемся преимущественно использовать объектные методы создания объектов (хотя бы для их изучения) и нам понадобятся функции преобразования координат.

Замечание

Насколько неудобно создание объектов только в МСК, знают VBA-программисты, столкнувшиеся с этой проблемой. Координаты объекта, созданного в МСК, можно преобразовать в ПСК (объект при этом "прыгнет" в другое место) с использованием метода `Utility.TranslateCoordinates` и трансформацией отрисованного примитива методом `TransformBy`, но это очень неудобно. Это же самое можно сделать и на LISP, но лучше мы будем заниматься предварительным преобразованием координат до создания объектов, а не тем же преобразованием и перемещением.

Координаты, извлекаемые из списков данных объектов, возвращаемых функцией `entget` по DXF-кодам 10 и 11, представлены в *системе координат объекта* (СКО).

Для точек, отрезков, трехмерных полилиний, трехмерных граней, сетей и размеров система координат объекта эквивалентна мировой, а для прочих примитивов СКО не совпадает с мировой и определяется направлением выдавливания объекта, содержащимся в группе с DXF-кодом 210. Для таких объектов требуется преобразование координат из СКО в МСК.

В соответствующих координатах должны передаваться аргументы и функции `entmake`. Это значительно осложняет работу по сравнению с использованием функции `command`, которая сама преобразует значения координат. При создании объектов с применением функции `entmake` или объектных методов мы должны правильно преобразовать координаты в зависимости от источника их получения. Допустим, при установленной ПСК мы будем "скалывать" точки полилинии с помощью выражения

```
(setq lst_pnt '())
(while (setq pnt (ru-get-point-or-exit  "" nil))
 (setq lst_pnt (cons pnt lst_pnt))
)
(setq lst_pnt(reverse lst_pnt))
```

а потом создадим полилинию-аналог с помощью функции, строящей полилинию объектными методами (пока без трансформации координат)

```
(ru-pline-add  lst_pnt nil 0 0 nil)
```

тогда новая полилиния окажется не там, где был прототип, а в другом месте. То же самое произойдет при использовании выражения (без трансформации координат)

```
(ru-pline-entmake lst_pnt nil 0 0 nil).
```

А вот при использовании "старушки" `command` в выражении

```
(command "_.PLINE")(mapcar 'command lst_pnt)(command "")
```

мы нарисуем полилинию-дубликат на том же самом месте.

Вспомните, в *главе 10* мы обсуждали достоинства и недостатки различных методов рисования и поставили *промежуточный* диагноз о том, что не всегда следует отказываться от использования функции `command`. С тех пор мы научились многому, но и значительно усложнили собственную работу. Если, как намечено, мы при рисовании будем отказываться от `command`, то должны и постоянно контролировать, откуда получены координатные данные. Конечно, в своих программах мы могли бы принудительно включать МСК, и иногда мы это будем делать, но только в тех случаях, когда это будет незаметно пользователю. Если же мы попытаемся устанавливать МСК во время ввода координат пользователем, то это будет плохим решением — пользователь мог установить ПСК, чтобы было удобнее вводить координаты с клавиатуры.

Как преобразовывать координаты

Преобразование координат точки осуществляется функцией `trans`.

Вызов `(trans point 0 1)` преобразует координаты точки из МСК в текущую ПСК, а `(trans point 1 0)` — из текущей ПСК в МСК.

А что произойдет, если попытаться повторно преобразовать координаты точки? Может быть ситуация, когда точку, имеющую мировые координаты, мы попытаемся повторно трансформировать. Проверим это экспериментально. Установим ПСК с началом координат в точке 10 000, 10 000 относительно мировой.

```
Command: UCS
Enter an option [New/Move/orthoGraphic/Prev/Restore/Save/Del/Apply/?/World]
<World>: m
Specify new origin point or [Zdepth]<0,0,0>: 10000,10000
```

Преобразуем точку в МСК.

```
Command: (trans (list 0 0) 1 0)
(10000.0 10000.0 0.0)
```

Мы получили правильное начало ПСК в МСК. Преобразуем точку МСК в ПСК.

```
Command: (trans (list 10000.0 10000.0 0.0) 0 1)
(0.0 0.0 0.0)
```

Вновь получено правильное значение. Попробуем вариант с `get`-функцией.

```
Command: (trans (getpoint "\nУкажи точку: ") 1 0)
Укажи точку: 0,0
(10000.0 10000.0 0.0)
```

Теперь изменим систему координат на мировую.

```
Command: ucs
Current ucs name:  *NO NAME*
Enter an option [New/Move/orthoGraphic/Prev/Restore/Save/Del/Apply/?/World]
<World>:W
```

Моделируем ситуацию с трансформацией точки с МСК-координатами в МСК.

```
Command: (trans (list 0 0) 1 0)
(0.0 0.0 0.0)
```

Координаты не преобразованы, но возвращена трехмерная точка (признак работы функции `trans`).

```
Command: (trans (list 10000.0 10000.0 0.0) 1 0)
(10000.0 10000.0 0.0)
```

Попробуем вариант с `get`-функцией.

```
Command: (trans (getpoint "\nУкажи точку: ") 1 0)
Укажи точку: 0,0
(0.0 0.0 0.0)
```

Таким образом мы *думаем*, что лишнее преобразование координат точек в МСК не вредит. Но, рассуждая таким образом, мы допускаем грубейшую ошибку — координаты точек в функцию `trans` мы передавали в виде списков, т. е. констант. Теперь попробуем передавать координаты через переменные. Установим ПСК.

```
Command: (setq pnt (list 0 0))
(0 0)
Command: (setq pnt (trans pnt 1 0))
(10000.0 10000.0 0.0)
```

Пока правильно. Повторим преобразование.

```
Command: (setq pnt (trans pnt 1 0))
(20000.0 20000.0 0.0)
```

Ошибка! И не удивительно — мы повторно преобразуем уже трансформированное значение, хранящееся в переменной. Попробуем с `get`-функцией.

```
Command: (setq pnt_get (trans (getpoint "\nУкажи точку: ") 1 0))
Укажи точку: 0,0
(10000.0 10000.0 0.0)
Command: (setq pnt_get (trans pnt_get 1 0))
(20000.0 20000.0 0.0)
Command: (setq pnt_get (trans pnt_get 1 0))
(30000.0 30000.0 0.0)
```

Как видим, ошибка нарастает. Читатели сами могут продолжить эксперименты и сделать выводы. Мы спешить с этим не будем.

Где трансформировать координаты

Итак, мы умеем трансформировать координаты и знаем, что для функции `command` всегда требуются координаты в ПСК, а для `entmake` и объектных функций — всегда в МСК. Трансформировать координаты можно в программе или в функции рисования, непосредственно перед их использованием. Хотелось бы сосредоточить создание объектов и требуемые трансформации в специальных функциях и нигде, кроме этих функций, непосредственно не рисовать объекты. Идея эта очень заманчива — мы располагаем сотнями программ, в которых рисование осуществлялось с помощью функции `command`, а координаты передавались "как есть". Все эти программы требуют серьезной переработки, и если мы будем в них еще и анализировать источники возникновения значений координат, то непременно запутаемся.

Однако если сделать преобразование непосредственно перед рисованием, например:

```
(vla-addline (ru-obj-active-space)
(vlax-3d-point (trans start_pnt 1 0))
(vlax-3d-point (trans end_pnt 1 0)))
```

то может оказаться, что это *только мы думаем*, что при таком варианте отрезок будет нарисован на нужном месте независимо от того, в какой системе координат были получены точки. На самом же деле могли произойти предварительные преобразования и значения переменных могут быть неправильными уже на входе в функцию рисования.

Ошибки лучше предотвращать в местах их возможного возникновения, поэтому разберемся, откуда же могут попадать в программы координаты точек. Таких источников немного:

- функции `getpoint` и `getcorner`, возвращающие координаты в *текущей ПСК*;
- функция `entget`, возвращающая координаты в МСК;
- функции `entsel` и `nentsel`, возвращающие точку указания примитива в *текущей ПСК*;

- объектные функции, возвращающие координатные свойства объектов в МСК;
- различные файлы с координатами.

Вариант с файлами самый непредсказуемый, но создание и чтение таких файлов — штучная работа и решение о трансформации координат должно приниматься индивидуально, хотя наилучшим вариантом было бы сохранение координат в файлах только в МСК.

Мы выделили источники возможных координатных ошибок, и видим, что нам повезло[1] — именно стандартные функции `getpoint`, `getcorner`, `entsel` и `nentsel` мы заменяем на собственные (`ru-get-point-xxx`, `ru-get-corner-required`, `ru-get-entsel-xxx`, `ru-get-nentsel`), да и сами они, в основном, используют низкоуровневые функции `_ru-get-with-default` и `_ru-get-ent-default`. Стандартные функции мы все равно везде заменяем на собственные, поэтому легко можем локализовать преобразования координат в МСК только на выходе из наших функций ввода. Низкоуровневую функцию `_ru-get-with-default` мы не будем модифицировать, т. к. она довольно часто применяется самостоятельно, а функции, связанные с передачей точек, изменим, например, так:

```
(defun ru-get-point-required (message base_point)
 (trans (_ru-get-with-default message nil 'getpoint 1 nil
   (if (ru-is-point base_point) (trans base_point 0 1) base_point))
   1 0)
)
```

или так (если возвращена точка, а не ключевое слово, точка трансформируется в МСК):

```
(setq first_point (_ru-get-with-default (strcat "Dist=" (rtos dist)
  ", Count=" (itoa count) ". Точка начала ряда") "Выход" 'getpoint
  nil "Dist Count" nil))
(if (ru-is-point first_point) (setq first_point (trans first_point 1 0)))
```

У `get`-функций есть одна проблема — некоторые из них принимают в числе аргументов базовую точку и значение по умолчанию. Базовая точка используется для отрисовки "резиновой нити", в какой системе она передается для *пользователя* значения не имеет, важно, чтобы нить тянулась из правильного места — для этого и преобразовываем базовые точки в ПСК. Значение по умолчанию отображается в командной строке и надо, чтобы оно отображалось в активной ПСК — для этого преобразовываем точки по умолчанию в ПСК. Если уж пользователь знает про ПСК[2] и сознательно их использует, то он и должен иметь возможность ввести данные именно в ПСК, например, вводом с клавиатуры.

Итак, сформулируем правила использования координат:

- во всех функциях и программах используются координаты точек в мировой системе координат;
- все наши функции ввода принимают и возвращают точки в мировой системе координат, позволяя при этом ввод координат пользователем в текущей системе координат;

---

[1] Мы сами подготовили это "везение", когда приняли решение о разработке таких функций.

[2] Огромное количество "теток", "чайников" и "ламеров" пребывают в счастливом неведении о существовании пользовательских систем координат.

- преобразование из мировой системы координат в пользовательскую производится только в те моменты, когда пользователь может выполнить клавиатурный ввод (например, подтвердить или изменить значение координат точки по умолчанию);
- в определенных случаях применения функций `command` и `vl-cmdf` производятся частные преобразования координат точек в текущую систему координат.

Сразу введем и правило для углов — *в функции рисования значения углов передаются в радианах*. Со старыми программами в этом отношении проще — от функции `command`, для которой производились преобразования в десятичные градусы, мы будем избавляться, а вычисления все равно выполнялись в радианах. Стандартные `get`-функции возвращают углы в радианах, хотя требуют ручного ввода углов в текущих единицах измерения углов.

И, наконец, самое главное правило — *разработчики должны обязательно испытывать свои программы при работе в пользовательских системах координат*.

Замечание

Предлагаем читателям немедленно испытать свои программы на работоспособность в ПСК. Если все они будут работать правильно, то одно из двух — или использована "старинная" методика с функцией `command` или вы исключительно предусмотрительны, знали все без наших советов и дальнейшее чтение книги для вас бесполезная трата времени.

Проблему с координатами мы должны были бы рассмотреть гораздо раньше — в *главе 10*. Конечно, мы о ней знали, но часто уклонялись от решения путем временной установки МСК. Удивительно, но несколько очень опытных программистов, с которыми мы консультировались, придерживались такой же политики — делать вид, что неприятностей не существует. Но если уж мы взялись за написание книги, то решили "добить" вопрос до конца. Просим читателей не удивляться, если в ранее опубликованных исходных текстах будут обнаружены нарушения провозглашенных правил — не все удастся исправить из-за особенностей издательской технологии. Но на компакт-диске, формируемом в последнюю очередь, будут правильные[1] тексты.

Результат работы функций рисования

Сразу решим вопрос и с результатами работы функций рисования. Желательно, чтобы любая функция возвращала какой-то результат, даже если он никак не используется. Для того чтобы функция "ничего не возвращала", последней вызываемой функцией должна быть `princ` без аргументов — так следует завершать высокоуровневые функции-программы, лучшим результатом работы которых будет тихое завершение без лишних сообщений[2]. Низкоуровневые функции вернут результат последней выполненной вложенной функции и надо продумывать, что именно вернется — иногда для возврата требуемого результата придется подставлять значение переменной.

Когда рисование осуществлялось исключительно с помощью `command`, использовать результаты ее работы не имело смысла — она всегда возвращала `nil`. При рисовании

[1] Или "правильные" в нашем сегодняшнем понимании — ничто не вечно, особенно в программировании.

[2] Альтернативные варианты — использование функций `prin1` и `print` без аргументов.

объектными методами мы можем вернуть созданный VLA-объект или `nil` при неудаче — это было бы логично и красиво. Но примитивы мы можем создавать еще и через `command`, и через `entmake`, и не можем раз и навсегда принять решение в пользу одного метода. Идеально было бы иметь три низкоуровневых функции рисования примитива, а вызывать в программах среднеуровневую функцию, выбирающую метод рисования в зависимости от какой-то глобальной переменной. Пример такого решения имеется в учебном пособии по Visual LISP, где используется три варианта функции рисования круга.

```
(defun gp:Create_activeX_Circle (center radius)
 (vla-addCircle *ModelSpace* (vlax-3d-point center) radius)
)
(defun gp:Create_entmake_Circle (center radius)
 (entmake (list (cons 0 "CIRCLE") (cons 10 center) (cons 40 radius)))
 (vlax-ename->vla-object (entlast))
)
(defun gp:Create_command_Circle (center radius)
 (command "_.CIRCLE" center radius)
 (vlax-ename->vla-object (entlast))
)
```

Первая функция строит круг с помощью функции ActiveX и возвращает VLA-объект, вторая строит круг посредством функции `entmake` и возвращает имя объекта, преобразованное в VLA-объект, а третья функция строит круг с помощью функции `command` и также возвращает имя объекта, преобразованное в VLA-объект. Выбор функции рисования осуществляется в зависимости от значения глобальной переменной.

```
(setq ObjectCreationFunction
 (cond
  ((equal ObjectCreationStyle "ACTIVEX") gp:Create_activeX_Circle)
  ((equal ObjectCreationStyle "ENTMAKE") gp:Create_entmake_Circle)
  ((equal ObjectCreationStyle "COMMAND") gp:Create_command_Circle)
  (T (alert (strcat " Неверное значение ObjectCreationStyle"
              "\nObjectCreationStyle установлен в ACTIVEX"))
          (setq ObjectCreationStyle "ACTIVEX")
          gp:Create_activeX_Circle[1]
  )
))
```

Мы пишем не учебное пособие, но не исключаем, что иногда будем менять методику рисования, не изменяя код множества программ. Фирма Autodesk дала хороший пример, предусмотрев преобразование результатов разных функций в единый тип данных — VLA-объект. Это разумно, т. к. прежде мы, в лучшем случае, получали имя примитива через `(entlast)`, а VLA-объект, при необходимости, мы сможем передать в другую функцию. Именно так мы и постараемся поступать[2], хотя возможны и другие варианты — возврат примитивов, последних точек и т. п.

---

[1] Исправляем ошибку программистов Autodesk — без добавления этой строчки переменной `ObjectCreationFunction` будет присвоено значение переменной `ObjectCreationStyle`, что приведет к последующей ошибке. Это пример того, как не проверяются все ситуации.

[2] По крайней мере в тех случаях, когда прежде результат работы функции никак не использовался.

Реализация координатных функций

Хотя мы решили локализовать преобразования координат, и можем осуществлять их только с помощью функции `trans`, разработаем небольшой набор простых функций для работы с системами координат. Начнем с проверки, является ли активная система координат не мировой (листинг 22.14).

Листинг 22.14. Функция `ru-ucs-is-ucs`

```
(defun ru-ucs-is-ucs ()
    (zerop (getvar "WORLDUCS"))
)
```

Функции преобразования списков точек приведены в листингах 22.15 и 22.16.

Листинг 22.15. Функция `ru-ucs-trans-list-points-wcs-ucs`

```
(defun ru-ucs-trans-list-points-wcs-ucs (points)
    (mapcar (function (lambda (point) (trans point 0 1))) points)
)
```

Листинг 22.16. Функция `ru-ucs-trans-list-points-ucs-wcs`

```
(defun ru-ucs-trans-list-points-ucs-wcs (points)
    (mapcar (function (lambda (point) (trans point 1 0))) points)
)
```

Мы очень часто будем использовать типовую форму списка координат полилинии, в котором первым элементом является список точек, а вторым — флаг замкнутости. Для упрощения трансформации таких списков введем еще пару функций (листинги 22.17 и 22.18).

Листинг 22.17. Функция `ru-ucs-trans-vertex-wcs-ucs`

```
(defun ru-ucs-trans-vertex-wcs-ucs (list_vert)
 (if (ru-ucs-is-ucs)
   (list (ru-ucs-trans-list-points-wcs-ucs (car list_vert))
         (cadr list_vert)
   )
    list_vert
 )
)
```

Листинг 22.18. Функция `ru-ucs-trans-vertex-ucs-wcs`

```
(defun ru-ucs-trans-vertex-ucs-wcs (list_vert)
  (if (ru-ucs-is-ucs)
      list_vert
        (list (ru-ucs-trans-list-points-ucs-wcs (car list_vert))
```

```
                (cadr list_vert)
        )
  )
)
```

Некоторые команды не работают, если ось Z текущей ПСК не параллельна оси Z МСК, поэтому потребуется функция, приведенная в листинге 22.19.

Листинг 22.19. Функция `ru-ucs-z-is-parallel-wcs`

```
(defun ru-ucs-z-is-parallel-wcs ()
  (= 1.0 (caddr (trans '(0.0 0.0 1.0) 1 0)))
)
```

Выполним комплексную проверку. Установим пользовательскую систему координат, нарисуем полилинию, у которой первая вершина находится в начале ПСК, и получим

```
Command: (ru-pline-list-vertex (car (entsel)))
Select object: (((10000.0 10000.0) (20000.0 10000.0) (20000.0 15000.0) (10000.0
15000.0)) nil).
```

Функция вернула список вершин в МСК. Теперь выполним преобразования. Напоминаем, текущая система — пользовательская.

```
Command: (ru-ucs-trans-vertex-ucs-wcs (ru-pline-list-vertex (car (entsel))))
Select object: (((10000.0 10000.0) (20000.0 10000.0) (20000.0 15000.0)
(10000.0 15000.0)) nil)
```

Возвращены непреобразованные координаты в МСК (видно по двумерности точек). А зачем транслировать, если и так в МСК?

```
Command: (ru-ucs-trans-vertex-wcs-ucs (ru-pline-list-vertex (car (entsel))))
Select object: (((0.0 0.0 0.0) (10000.0 0.0 0.0) (10000.0 5000.0 0.0)
(0.0 5000.0 0.0)) nil)
```

Координаты правильно преобразованы в ПСК. Теперь устанавливаем МСК и повторяем тесты.

```
Command: (ru-ucs-trans-vertex-ucs-wcs (ru-pline-list-vertex (car (entsel))))
Select object: (((10000.0 10000.0 0.0) (20000.0 10000.0 0.0) (20000.0 15000.0 0.0)
(10000.0 15000.0 0.0)) nil)
```

Координаты правильно преобразованы в МСК.

```
(ru-ucs-trans-vertex-wcs-ucs (ru-pline-list-vertex (car (entsel))))
Select object: (((10000.0 10000.0) (20000.0 10000.0) (20000.0 15000.0)
(10000.0 15000.0)) nil)
```

Возвращены непреобразованные координаты в МСК.

Извлечение списка координат вершин

Теперь разберемся, как работает функция `ru-pline-list-vertex` (листинг 22.20). Она возвращает упомянутый выше типовой список координат вершин с флагом замкнутости для полилиний и отрезков.

Листинг 22.20. Функция `ru-pline-list-vertex`

```
(defun ru-pline-list-vertex (ent / list_vertex tmp_ent type_ent)
 (setq tmp_ent ent ent (entget ent) type_ent (cdr (assoc 0 ent)))
 (cond
  ((= "LWPOLYLINE" type_ent)
   (list (ru-list-massoc 10 ent) (= 1 (logand 1 (cdr (assoc 70 ent))))))
  ((= "POLYLINE" type_ent)
    (reverse
      (cons (= 1 (logand 1 (cdr (assoc 70 ent))))
            (while (and (setq tmp_ent (entnext tmp_ent))
                        (/= (cdr (assoc 0 (setq ent (entget tmp_ent))))
                          "SEQEND"));_ end of and
                (setq list_vertex
                     (cons (cdr (assoc 10 ent)) list_vertex))
            );_ end of while
    );_ end of cons
   );_ end of reverse
 )
  ((= "LINE" type_ent)
  (list (cons (cdr (assoc 10 ent)) (cons (cdr (assoc 11 ent)) nil)) NIL))
  (T nil)
);_ end of cond
);_ end of defun
```

Теперь мы готовы разрабатывать *надежные* функции рисования. Из многообразия примитивов AutoCAD для нашей системы потребуется не так уж много — блоки, отрезки, полилинии, круги, штриховки. Начнем с любимых блоков.

Как эффективно использовать блоки

Очень много программ будут использовать блоки. Основные направления применения блоков мы обсуждали в *главе 6*. Просмотрите еще раз эту главу, чтобы освежить в памяти, какие виды блоков и как мы договорились использовать.

Пока мы занимаемся программированием, группа "продвинутых" пользователей занимается подготовкой блоков и формированием библиотек блоков (кстати, надо проверить, чем они там на самом деле занимаются). Теперь нам предстоит реализовать работу с блоками на функциональном уровне.

Чаще всего блоки мы будем вставлять, иногда — создавать "на лету" и переопределять. Вставка блоков должна происходить с максимальными удобствами для пользователя. В принципе, он может вообще ничего не знать о блоках, не должен высчитывать масштаб вставки и углы поворота блоков. Программа должна сама знать, как масштабировать блок и что спрашивать у пользователя — вообще ничего, только точку вставки или еще и угол поворота.

Для того чтобы блок можно было вставить, он должен быть *определен* в рисунке. Вначале в рисунке нет блоков[1], они хранятся в библиотеках блоков или в отдельных

[1] Здесь мы не учитываем блоки, возможно хранящиеся в шаблонах и соответственно оказывающиеся в новых рисунках, созданных на основе этих шаблонов.

файлах, следовательно, блок прежде всего нужно внедрить в рисунок из внешнего файла. Определять местонахождение файлов мы будем с помощью простых функций, подобных приведенной в листинге 22.21.

Листинг 22.21. Функция `ru-file-block-lib`

```
(defun ru-file-block-lib (name)
;; Возвращает полное имя файла в каталоге библиотек блоков
    (strcat (ru-dirs-get-all-users) "block-lib\\" name)
)
```

Большинство наших блоков сосредоточено в библиотеках блоков. Для использования блока из библиотеки мы должны внедрить всю библиотеку. Так как неиспользуемые блоки могут быть удалены, необходимо каждый раз проверять наличие определения нужного блока, а при его отсутствии — внедрять всю библиотеку (листинг 22.22). Обратите внимание, что вставка библиотечного блока производится функцией `command` и, в то время, когда система AutoCAD должна задать вопрос о точке вставки, мы вновь вызываем функцию `command` без аргументов. Такой вызов аналогичен нажатию клавиши <Esc> и прерывает вставку. В результате библиотечный блок (со всеми вложенными в него "блочишками") оказывается внедренным в рисунок, но не вставленным.

Листинг 22.22. Функция `ru-block-lib-insert`

```
(defun ru-block-lib-insert (block_lib_name block_name / block_lib_file_name)
;|-----------------------------------------------------------------------
;;; Внедрение библиотечного блока block_lib_name, в котором живет блок block_name
Пример
(ru-block-lib-insert "R2000_00000000" "НОМЕР_УЗЛА") > T
-----------------------------------------------------------------------|;
  (if (tblsearch "block"  block_name)
    T
    (progn
      (setq block_lib_file_name (ru-file-block-lib block_lib_name))
      (if (findfile (strcat block_lib_file_name ".DWG"))
       (progn
         (princ "\nПодождите...")
         (setvar "CMDECHO" 0)
         (command "_.INSERT" block_lib_file_name)
         (command); прерываем вставку, блок невидимый
         (if (tblsearch "block"  block_name)
         ;;Проверим блок после внедрения
           T
          (progn
            (ru-msg-alert       (strcat "ОШИБКА!\n Блок\n" block_name
             "\nотсутствует в библиотеке\n" block_lib_file_name
              "!\nОбратитесь к администратору системы!"))
           nil
       );_ end of progn
      );_ end of if
     );_ end of progn
```

```
    (progn
      (ru-msg-info (strcat "Не найден библиотечный блок\n"
                    block_lib_file_name))
       nil
    );_ end of progn
   );_ end of if
  );_ end of progn
 );_ end of if
)
```

Вставка блока

Разработаем функцию для "тихой" вставки блока в заданную точку с заданными масштабами и поворотом. Функцией (command "_.INSERT" ...) это можно сделать в одну строчку, но надо учитывать возможные атрибуты, подавлять их запросы, предотвращать влияние включенной объектной привязки и т. п. Будем переходить на объектные методы. Ненадежную объектную вставку блока также можно реализовать в одной строчке, но мы применяем дополнительные приемы:

- трансформирование координат точки вставки в мировую систему;
- ловушку ошибок, которая позволяет, при сбое по любой причине, корректно завершить функцию с сообщением о причине ошибки.

При удачной вставке функция вернет VLA-объект блока, при неудаче — nil. В момент вставки диалоговое окно редактирования атрибутов не выводится. Текст функции приведен в листинге 22.23.

Листинг 22.23. Функция ru-block-insert-obj

```
(defun ru-block-insert-obj (block_name ins_pnt x_scale y_scale z_scale rotation)
;;; Вставка блока. Возвращает VLA-object
  (ru-error-catch
    (function (lambda ()
       (vla-insertblock
        (ru-obj-active-space) (vlax-3d-point (trans ins_pnt 1 0))
         block_name x_scale y_scale z_scale rotation))
   );_ end of function
    (function
     (lambda (x) (princ (strcat "\nОШИБКА RU-BLOCK-INSERT-OBJ " x)) nil))
 );_ end of ru-error-catch
)
```

Вставка блоков, у которых могут быть атрибуты, производится функцией ru-block-insert-attedit (листинг 22.24)

Листинг 22.24. Функция ru-block-insert-attedit

```
(defun ru-block-insert-attedit (block_name ins_pnt x_scale y_scale z_scale rotation
/ obj)
(if (setq obj (ru-block-insert-obj block_name ins_pnt x_scale y_scale
                z_scale rotation))
```

```
  (progn
   (if (= (vla-get-hasattributes obj) :vlax-true)
    (vl-cmdf "_.DDATTE" (vlax-vla-object->ename obj))
  ) obj) nil)
)
```

Многократная вставка блока

Одной из самых востребованных в нашей системе будет функция многократной вставки блока. Вставить блок можно несколькими способами, но нам требуется одновременное выполнение следующих условий:

- блок требуется вставлять *много раз*, в цикле, с заданными масштабами и, возможно, углом поворота;
- масштабы вставки могут быть разными по всем осям, в том числе отрицательными;
- при *каждом* запросе точки вставки, *включая первый*, изображение блока должно быть привязано к курсору, т. е. пользователь должен видеть, что он вставляет и как это впишется в обстановку;
- пользователь должен иметь возможность отказаться от вставки (включая первый раз) с нормальным завершением функции и основной программы;
- пользователь должен иметь возможность указать точку вставки и угол поворота *любыми* стандартными способами (мышью или вводом с клавиатуры);
- блок может иметь атрибуты любых типов (скрытые, постоянные, контролируемые, установленные), которые, возможно, надо заполнять с помощью диалогового окна;
- пользователь должен иметь возможность пользоваться объектной привязкой и выполнять прозрачные команды;
- работа функции не должна завершаться нажатием клавиши <Esc> в качестве основного средства прерывания вставки (хотя возможно в качестве дополнения к штатному выходу при пустом вводе; при этом не должна прерываться основная программа);
- функция должна работать "чисто" — во время работы за курсором не должны тянуться "резиновые" нити, а при любом, в том числе аварийном завершении, не должны оставаться "трупы" временных объектов и прочий мусор;
- функция должна работать очень надежно, т. к. является базой для множества иных функций.

На первый взгляд кажется, что задачу решить легко. Вставить висящий на курсоре масштабированный и повернутый блок можно так:

```
(command "_.INSERT" block_name
  "_PX" x_scale "_PY" y_scale "_PRotate" ang pause
  x_scale y_scale  ang)
```

Однако так можно вставить блок один раз (точка вставки вводится во время ожидания pause), или организовать цикл путем запроса на продолжение после, а не до первой вставки.

Замечание

Лучше всего, если бы возможность такой работы была бы реализована в самой системе AutoCAD, как это сделано в команде DONUT (КОЛЬЦО) — после ввода диаметров "баранка" висит на курсоре и ее можно много раз вставлять в нужные точки, завершая цикл пустым вводом. Увы, такое простое в реализации усовершенствование, видимо, не появится. Зато добавляются все новые усложнения не первой необходимости.

Приводим текст этой функции с трудно запоминаемым названием (листинг 22.25) с подробными комментариями.

Листинг 22.25. Функция `ru-block-multi-insert-scaled-rotated-or-angleask`

```
(defun ru-block-multi-insert-scaled-rotated-or-angleask (block_name
x_scale y_scale block_angle / curr_layer do_insert ins_point
is_attrib sub_ent tmp_angle tmp_block tmp_ent unnamed_block put_point
exp_block)
;|-----------------------------------------------------------------
Множественная вставка блока, висящего на курсоре
Аргументы:
block_name - имя блока, определенного в рисунке
x_scale - масштаб по X
y_scale - масштаб по Y
block_angle - угол поворота
-----------------------------------------------------------------|;
;|-----------------------------------------------------------------
Проверяем, не установлен ли перспективный вид - в нем не будет работать команда
CHANGE
-----------------------------------------------------------------|;
 (if (ru-ucs-z-is-parallel-wcs)
  (progn
   (setq do_insert  t
;|-----------------------------------------------------------------
Временная точка вставки за пределами видимой части экрана
-----------------------------------------------------------------|;
    ins_point  (list
     (+ (car (getvar "VSMAX")) (* 2 (max x_scale y_scale)))
     (+ (cadr (getvar "VSMAX")) (* 2 (max x_scale y_scale)))
      0.0)
;|-----------------------------------------------------------------
Временный угол поворота блока. Нужен, если в аргументах поворот задан NIL
(признак необходимости запроса угла поворота)
-----------------------------------------------------------------|;
    tmp_angle  (if block_angle block_angle 0)
    curr_layer (getvar "CLAYER"))
;|-----------------------------------------------------------------
Вставляем временный блок за пределы видимой части экрана
-----------------------------------------------------------------|;
   (if (setq tmp_block (ru-block-insert-obj block_name
                    ins_point x_scale y_scale 1.0 tmp_angle))
```

```
    (progn
;|-------------------------------------------------------------------
Создаем из временного блока временный анонимный блок, чтобы обеспечить вставку
с масштабными коэффициентами с одинаковым знаком.
-------------------------------------------------------------------|;
      (setq tmp_block (vlax-vla-object->ename tmp_block)
                      tmp_ent   tmp_block)
      (entmake (list (cons 0 "BLOCK") (cons 100 "AcDbEntity")
         (cons 100 "AcDbBlockBegin") (cons 2 "*U0") (cons 8 curr_layer)
         (cons 70 1) (cons 62 256) (cons 10 ins_point)))
      (entmake (entget tmp_ent))
;|-------------------------------------------------------------------
Создаем для временного блока атрибуты, имеющиеся в основном блоке
-------------------------------------------------------------------|;
      (while (setq tmp_ent (entnext tmp_ent))
       (if
         (= (cdr (assoc 0 (setq sub_ent (entget tmp_ent)))) "ATTRIB")
           (setq is_attrib T)) (entmake sub_ent))
;|-------------------------------------------------------------------
Создаем анонимный блок. Чтобы использовать его как двойника настоящего для процесса
вставки и применения команды 'CHANGE'.
-------------------------------------------------------------------|;
      (setq unnamed_block (entmake (list (cons 0 "ENDBLK")
          (cons 100 "AcDbEntity") (cons 100 "AcDbBlockEnd")
                                          (cons 8 curr_layer))))
;|-------------------------------------------------------------------
Удаляем временный блок
-------------------------------------------------------------------|;
      (entdel tmp_block)
   );_ end of progn
    (setq do_insert nil)
  );_ end of if
;|-------------------------------------------------------------------
Начинаем цикл вставки. Вставляем анонимный блок во временную точку вставки
-------------------------------------------------------------------|;
    (while do_insert
     (setq tmp_block (vlax-vla-object->ename
       (ru-block-insert-obj  unnamed_block ins_point 1.0 1.0 1.0 0.0)))
;|-------------------------------------------------------------------
Имитируем вставку. Запрашиваем точку вставки, но на самом деле используем команду
CHANGE. В этот момент изображение блока висит на курсоре
-------------------------------------------------------------------|;
     (princ "\nТочка вставки <Хватит!>: ")
     (if (or (not (setq do_insert
;|-------------------------------------------------------------------
Используем особенность функции vl-cmdf - возможность возврата T или NIL
-------------------------------------------------------------------|;
            (vl-cmdf "_.CHANGE" tmp_block "" "" pause ""))
;|-------------------------------------------------------------------
Проверка на совпадение измененной "точки вставки" с прежним положением блока.
При нажатии <Enter> команда CHANGE оставит блок в прежней точке.
```

```
Условие совпадения точек является проверкой нажатия <Enter>. При нажатии <Esc>
функция vl-cmdf вернет NIL. Продолжение процессов "вставки" будет только при
указании точки, но не при нажатии клавиш <Enter> или <Esc>. Обратите внимание -
совпадение координат точек проверяется функцией equal с указанием допуска,
а не функцией =. Об этом мы писали в главе 10.
----------------------------------------------------------------------|;
          (equal (setq put_point  (cdr (assoc 10 (entget tmp_block))))
                 ins_point 1e-6))
     (setq do_insert nil)
    (progn
;|----------------------------------------------------------------------
Если угол поворота был в аргументах задан как NIL, аналогичным образом
выполняется визуальный поворот блока
----------------------------------------------------------------------|;
      (if (not block_angle)
       (progn
        (princ "\nУгол поворота <0>: ")
        (setq do_insert (vl-cmdf "_.CHANGE" tmp_block "" "" "" pause)))
     (if do_insert
      (progn
;|----------------------------------------------------------------------
 Адаптация к 2004-му. Учтена особенность (явный глюк) 2004-го, когда при
расчленении безымянного блока он сбрасывает результат в точку вставки безымянного
блока + точку, использованную при его определении.
----------------------------------------------------------------------|;
     (setq exp_block
        (car
          (vlax-safearray->list (vlax-variant-value
            (vla-explode (vlax-ename->vla-object tmp_block)))
         )))
;|----------------------------------------------------------------------
Выполняется редактирование атрибутов, если они есть
----------------------------------------------------------------------|;
  (if is_attrib (vl-cmdf "_.DDATTE" (entlast)))
   (vla-put-insertionpoint exp_block (vlax-3d-point put_point))
  ))))
;|----------------------------------------------------------------------
Удаляем временный блок.
----------------------------------------------------------------------|;
   (entdel tmp_block)))
    (alert (strcat "\nОсь Z текущей ПСК не параллельна оси Z МСК."
                   "\nВыполение команды прекращено."))
  )
  (princ)
)
```

Мы разрабатывали множество вариантов этой функции, начиная с системы AutoCAD R10. Все они оказывались или громоздкими, или ненадежными, причем надежность была обратно пропорциональна "изящности". Несколько раз мы публиковали условия задачи с известными решениями и возникающими проблемами в

Интернете и получали массу дельных (и не очень дельных) советов[1]. Определенную сумятицу вносило и совершенствование AutoCAD. Например, упростило работу введение функции `vl-cmdf`, но очень усложняло разное поведение одних и тех же функций в разных версиях AutoCAD. Последний (надеемся, что последний) сюрприз преподнесла система AutoCAD 2004 — функция, надежно работающая в AutoCAD 2002, вдруг "сходила с ума" в этой версии.

Первое впечатление было такое, что функция стала удалять окончательную вставку блока, вместо того, чтобы удалить временный анонимный блок, создаваемый в ходе ее выполнения. Как показало исследование, на самом деле происходило совсем другое — удалялся, по-прежнему, временный анонимный блок, но вот окончательная вставка блока оказывалась примерно в той точке, в которую вставлялся этот анонимный блок. Происходило это по той причине, что функция `vla-explode` в версиях AutoCAD 2000—2002 при расчленении анонимных блоков оставляет примитивы, получившиеся в результате расчленения на тех же местах, где они были в составе вставки, а в версии 2004 примитивы создаются со смещением, соответствующим точке вставки, которая использовалась при создании анонимного блока. Такая "модернизация" поведения функций от версии к версии явление крайне неприятное, совершенно недопустимое при профессиональном подходе, но, к сожалению, встречающееся[2].

Может возникнуть вполне закономерный вопрос — а зачем, собственно, создается временный анонимный блок из вставки пользовательского блока? На первый взгляд, процедура эта совершенно лишена смысла, но, как это часто бывает, причина снова кроется в ошибках системы AutoCAD — начиная с версии 2000, команда CHANGE (ИЗМЕНИТЬ) имеет ошибку, которая выражается в некорректном перемещении вставки блока с различными по знаку масштабными коэффициентами по разным осям. Для решения этой проблемы и был предусмотрен механизм создания временного анонимного блока из вставки пользовательского блока. При этом пользовательский блок вставляется с теми масштабными коэффициентами, которые заданы при вызове функции, но к этой вставке не применяется команда CHANGE (ИЗМЕНИТЬ), а анонимный блок, которым подменяется пользовательский блок, всегда используется со всеми коэффициентами, равными единице, соответственно, у команды CHANGE (ИЗМЕНИТЬ) не возникает повода проявить свой норов.

Фактически, к настоящему времени сложилась несколько странная ситуация — чуть ли не вся эта функция представляет собой сплошную ловушку для ошибок и различий поведения разных версий AutoCAD.

Мы не считаем приведенную функцию совершенной и надеемся, что кто-нибудь из более сообразительных, чем мы, разработчиков сумеет ее улучшить. Хотим только предостеречь от кажущихся очевидными, но ненадежных вариантов, например с использованием вставки из буфера обмена. Нам необходимо, чтобы выполнялись *все*, а не большинство условий задачи.

---

[1] Наше творческое сотрудничество с неоднократно упоминаемым в книге Петром Лоскутовым началось с усовершенствований этой функции. Именно он взял на себя эту грязную работу и несет этот "тяжкий крест" уже несколько лет.

[2] Может оказаться вполне достаточно модернизации поведения одной функции, чтобы доставить независимым разработчикам массу "удовольствия".

Различные способы вставки блоков

Имея базовые функции вставки блока, можно разработать множество вариантов их применения. Приведем некоторые. С давних времен мы используем функцию многократной вставки блока с автоматическим поворотом и масштабированием (листинг 22.26). Коэффициенты масштабирования и угол поворота определяются специальным кодом. Этот прием применяется для вставки одного и того же блока в десяти различных видах (обычно в аксонометрических схемах "под 45 градусов" сантехнических систем).

Листинг 22.26. Функция _ru-block-multi-insert-scaled-angle0

```
(defun _ru-block-multi-insert-scaled-angle0 (block_name ins_code x_scale y_scale /
x y angle_deg)
  (cond
    ((= ins_code 1) (setq x 1 y 1 angle_deg 0.0))
    ((= ins_code 2) (setq x 1 y 1 angle_deg 180.0))
    ((= ins_code 3) (setq x 1 y -1 angle_deg 135.0))
    ((= ins_code 4) (setq x -1 y 1 angle_deg 135.0))
    ((= ins_code 5) (setq x 1 y -1 angle_deg 90.0))
    ((= ins_code 6) (setq x -1 y 1 angle_deg 90.0))
    ((= ins_code 7) (setq x -1 y -1 angle_deg 135.0))
    ((= ins_code 8) (setq x 1 y 1 angle_deg 135.0))
    ((= ins_code 9) (setq x -1 y 1 angle_deg 0.0))
    ((= ins_code 10) (setq x 1 y -1 angle_deg 0.0))
    (T (princ "\nНеверный код вставки!\n")
     (setq x 1 y 1 angle_deg 0.0))
 )
  (ru-block-multi-insert-scaled-rotated-or-angleask block_name
    (ru-conv-millimeter-in-paper-to-unit (* x x_scale))
    (ru-conv-millimeter-in-paper-to-unit (* y y_scale))
    (ru-conv-deg-to-rad angle_deg))
  (princ)
)
```

Следующая подобная функция (листинг 22.27) вставляет блоки в плоские схемы с запросом угла поворота.

Листинг 22.27. Функция _ru-block-multi-insert-scaled-angleask

```
(defun _ru-block-multi-insert-scaled-angleask (block_name ins_code x_scale y_scale)
   (ru-block-multi-insert-scaled-angleask block_name
   (*(if (= ins_code 1) 1 -1) x_scale) y_scale)
)
```

Функция из листинга 22.28 вставляет блоки с углом поворота 0 градусов.

Листинг 22.28. Функция ru-block-multi-insert-scaled-angle0

```
(defun ru-block-multi-insert-scaled-angle0 (block_name x_scale y_scale)
  (ru-block-multi-insert-scaled-rotated-or-angleask block_name
```

```
    (ru-conv-millimeter-in-paper-to-unit x_scale)
    (ru-conv-millimeter-in-paper-to-unit y_scale) 0.0)
  (princ)
)
```

Функция (листинг 22.29) вставляет блоки с запросом угла поворота, но, в отличие от приведенной в листинге 22.27, не использует код масштабирования.

Листинг 22.29. Функция ru-block-multi-insert-scaled-angleask

```
(defun ru-block-multi-insert-scaled-angleask (block_name x_scale y_scale)
  (ru-block-multi-insert-scaled-rotated-or-angleask block_name
    (ru-conv-millimeter-in-paper-to-unit x_scale)
    (ru-conv-millimeter-in-paper-to-unit y_scale) nil)
  (princ)
)
```

Единичная вставка блоков

Теперь проработаем однократную визуальную вставку блока (листинг 22.30). Мы, без всяких затей, используем функцию `vl-cmdf`, т. к. нам требуется изображение блока на курсоре. С помощью опций команды мы предварительно масштабируем блок, а после вставки устанавливаем окончательный масштаб. Ловушка ошибок предотвращает сбой при нажатии клавиши <Esc> во время вставки. Эта функция используется для вставки блоков, имеющих такие размеры, которые должны быть на бумаге (условные знаки, таблицы).

Листинг 22.30. Функция ru-block-insert-scaled-ptask

```
(defun ru-block-insert-scaled-ptask (block_name x_scale y_scale / x y old_attdia
result)
  (setq old_attdia (getvar "ATTDIA")) (setvar "ATTDIA" 1)
  (setq x (ru-conv-millimeter-in-paper-to-unit x_scale)
        y (ru-conv-millimeter-in-paper-to-unit y_scale))
  (while (not (ru-error-catch (function (lambda ()
     (princ "\nТочка вставки: ")
     (if (vl-cmdf "_.INSERT" block_name "_PROTATE" 0 "_PXScale" x
            "_PYScale" y pause x  y 0)
             (setq result (entlast))))) nil))
 );_ end of while
 result
)
```

Часто требуется вставка изображений, которые нужно масштабировать и поворачивать визуально (деревья, люди). Для таких целей используется функция, приведенная в листинге 22.31. Ей передается имя блока, а далее имитируется работа стандартной команды.

Листинг 22.31. Функция ru-block-insert-mousescaleask-angleask

```
(defun ru-block-insert-mousescaleask-angleask (block_name / old_attdia old_echo)
  (setq old_attdia (getvar "ATTDIA") old_echo (getvar "CMDECHO"))
```

```
  (setvar "ATTDIA" 1) (setvar "CMDECHO" 0)
  (princ "\nТочка вставки: ") (vl-cmdf "_.INSERT" block_name)
  (setvar "CMDECHO" 1)
  (while (wcmatch (getvar "CMDNAMES") "*INSERT*") (command pause))
  (setvar "ATTDIA" old_attdia) (setvar "CMDECHO" old_echo)
  (princ)
)
```

Вставка блоков из файлов

Пока мы использовали для вставки блоки, определенные в рисунке. Теперь напишем функции, вставляющие блоки с загрузкой из библиотек или из отдельных файлов. Эти функции вызываются из макросов различных меню. Многократная вставка блока из библиотеки приведена в листинге 22.32.

Листинг 22.32. Функция `ru-block-insert-from-lib`

```
(defun ru-block-insert-from-lib (block_lib_name block_name ins_code x_scale y_scale
is_angle0)
 (ru-app-begin)
 (if (ru-block-lib-insert block_lib_name block_name)
   (if is_angle0
    (_ru-block-multi-insert-scaled-angle0
         block_name ins_code x_scale y_scale)
    (_ru-block-multi-insert-scaled-angleask
         block_name ins_code x_scale y_scale)))
  (ru-app-end)
)
```

Вставка блока из каталога таблиц показана в листинге 22.33 (имеется и специальная программа, рисующая таблицу с разграфкой). Так как файлы таблиц должны рисоваться в масштабе 1:1 с единицей миллиметры, предусмотрено соответствующее масштабирование.

Листинг 22.33. Функция `ru-block-insert-table`

```
(defun ru-block-insert-table (block_name / old_attdia scale)
  (ru-app-begin) (setq old_attdia (getvar "attdia")) (setvar "attdia" 1)
  (setq scale (ru-conv-millimeter-in-paper-to-unit 1))
  (while (not (ru-error-catch (function (lambda ()
   (princ "\nТочка вставки: ")
   (vl-cmdf "_.INSERT" (ru-file-table block_name) "_PROTATE" 0 "_PScale"
      scale pause scale scale 0))) nil)))
  (setvar "attdia" old_attdia) (ru-app-end) (princ)
)
```

Далее приведем еще несколько функций, вставляющих блоки разными ранее рассмотренными методами из файлов (листинги 22.34—22.36).

Листинг 22.34. Функция `ru-block-insert-from-lib-mousescaleask`

```
(defun ru-block-insert-from-lib-mousescaleask (block_lib_name block_name)
  (ru-app-begin)
  (if (ru-block-lib-insert block_lib_name block_name)
    (ru-block-insert-mousescaleask-angleask block_name))
  (ru-app-end)
)
```

В библиотеках содержатся и блоки, нарисованные в натуральную величину и с миллиметрами в качестве единиц рисунка — *блоки-изделия (см. главу 6)*. Для вставки таких блоков применяется функция, приведенная в листинге 22.35. Она выполняет соответствующее масштабирование с учетом единиц текущего рисунка. Это позволяет вставить, например, изображение автомобиля и на план здания, и на генплан.

Листинг 22.35. Функция `ru-block-insert-from-lib-scale1`

```
(defun ru-block-insert-from-lib-scale1 (block_lib_name block_name is_angle0)
 (ru-app-begin)
  (if (ru-block-lib-insert block_lib_name block_name)
    (ru-block-multi-insert-scaled-rotated-or-angleask block_name
      (ru-conv-millimeter-to-unit 1) (ru-conv-millimeter-to-unit 1)
      (if is_angle0 0 nil)
   );_ end of ru-block-multi-insert-scaled-rotated-or-angleask
 );_ end of if
  (ru-app-end)
)
```

Функция, показанная в листинге 22.36, вставляет файл из специального каталога блоков. Внутри имя файла передается с расширением, т. к. вставка объектным методом требует передачи или имени определенного блока, или полного имени файла.

Листинг 22.36. Функция `ru-block-insert-noscale-ptask-angleask`

```
(defun ru-block-insert-noscale-ptask-angleask
       (block_name / old_attdia old_echo)
  (ru-app-begin)
  (ru-block-multi-insert-scaled-rotated-or-angleask
    (ru-file-set-ext (ru-file-block block_name) ".dwg")
      (ru-conv-millimeter-to-unit 1) (ru-conv-millimeter-to-unit 1) nil)
  (ru-app-end) (princ)
)
```

А зачем нам столько мелких, похожих друг на друга функций для вставки блоков? Ну, это еще не много, на самом деле их больше. Излишняя номенклатура объясняется многообразием задач, решаемых в разных разделах проектов и с мелкими нюансами, предусмотренными для удобства работы. Действительно, форматы вызова некоторых функций можно было бы унифицировать, но они применяются уже давно, используются в тысячах макроопределений меню (в том числе созданных поль-

зователями), переделывать которые нет ни желания, ни времени. Кроме того, опыт показывает, что при любой переделке чего-то работающего непременно возникают новые ошибки, поэтому мы предпочитаем для новых задач создавать немного измененные варианты, не затрагивая интерфейсы работающих.

Читатели, не обремененные грузом прошлых разработок, могут сделать себе немного вариантов. Остальные напишете потом, когда "приспичит".

Изменение ширины линий в блоке

В *главе 6* мы упоминали про блоки с изменяемой шириной линий. Введение таких блоков позволило значительно сократить объем программирования для рисования изделий, когда одинаковые объекты на одном слое требуется изображать или тонкой, или "толстой" линией (например, существующие и проектируемые). Ранее для этого приходилось разрабатывать программы, рисующие объекты полилиниями с разной шириной. Введение веса линий не улучшило положение, да и не всегда удобно использовать вес линий. Разработаем функции, позволяющие получать изображение, подобное показанному на рис. 22.2, путем вставки одного блока.

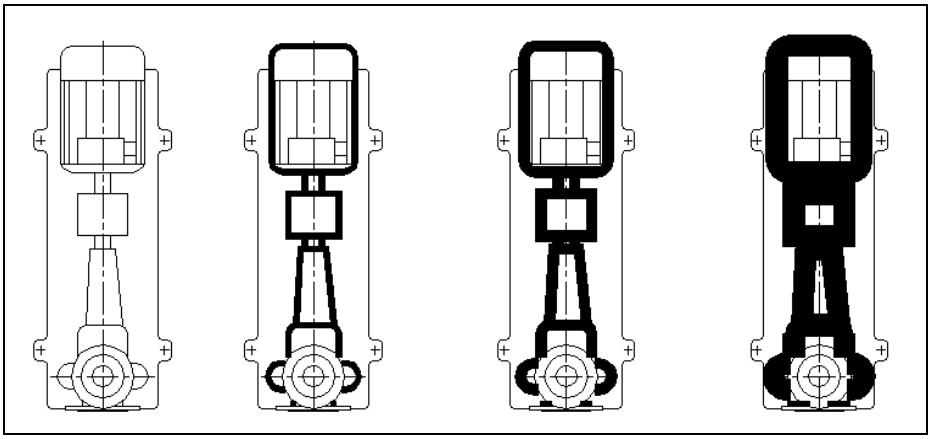


Рис. 22.2. Блоки, вставленные с изменением ширины линий

Блоки, для которых потребуется изменение ширины линий, должны быть подготовлены особым образом — линии, у которых требуется изменение ширины должны быть нарисованы компактными полилиниями (LWPOLYLINE), а остальные — любыми другими примитивами. Изменение ширины линий производится так:

1. Выполняется вставка блока в заданную точку с требуемыми масштабами и поворотом.
2. В диалоговом окне **Установка веса линии** (см. рис. 22.1) запрашивается требуемая ширина линий в блоке (выбирается вес линий, но использоваться он будет для определения ширины полилиний).
3. Вычисляется новое имя блока, включающее в себя имя исходного блока, ширину линий, масштаб и единицу измерения.

4. Если блок с таким именем определен в рисунке, вставленный прототип заменяется на блок с измененной шириной линий.
5. Если такой блок еще не определен, то он создается в рисунке с включением в определение блока примитивов исходного блока, но с измененной шириной полилиний, а затем вставка созданного блока заменяет прототип.

Реализуем алгоритм программно. Сначала разработаем главную функцию вставки блока (листинг 22.37).

Листинг 22.37. Функция ru-block-lw-insert-ptask-angleask

```
(defun ru-block-lw-insert-ptask-angleask (block_name scale / edata ent old_attdia)
 (setq old_attdia (getvar "ATTDIA")) (setvar "ATTDIA" 1)
 (while (not (ru-error-catch (function (lambda ()
   (princ "\nТочка вставки: ")
   (if (vl-cmdf "_.INSERT" block_name "_PROTATE" 0 "_PScale" scale
      pause scale scale)
     (progn (princ "\nУгол поворота<0>: ")
     (if (vl-cmdf pause)
     (progn
     (setq edata (entget (setq ent (entlast))))
     (if (= "INSERT" (cdr (assoc 0 edata)))
       (progn
;|-------------------------------------------------------------------
Выбираем в диалоговом окне ВЕС, но изменять будем ШИРИНУ.
Функция (ru-dlg-dcl-select-lw) не изменяет текущий вес линий в AutoCAD, а изменяет
результат, возвращаемый функцией (ru-lw-current)
-------------------------------------------------------------------|;
        (ru-dlg-dcl-select-lw)
        (ru-block-lw-change ent (cdr (assoc 2 edata))
             (cdr (assoc 10 edata)) (cdr (assoc 50 edata))
             (cdr (assoc 41 edata))))
        (ru-msg-info "Этот объект НЕ блок!\nНЕЛЬЗЯ преобразовать!"))
      )))))) nil)))
  (setvar "attdia" old_attdia) (princ)
)
```

Основная работа происходит в функции `ru-block-lw-change`, которой передается примитив вставки блока-прототипа и параметры вставки — имя блока, точка вставки, угол поворота и масштаб. Много лет мы применяли подобную функцию, в которой использовалась функция `command`, но теперь, из нездорового любопытства к "пакостям" объектной модели, все выполним объектными методами (листинг 22.38).

Листинг 22.38. Функция ru-block-lw-change

```
(defun ru-block-lw-change (ent name ins_pnt ang scale /)
;|-------------------------------------------------------------------
Вычисляем требуемую ширину полилиний. Для этого извлекаем ВЕС линий своей функцией
ru-lw-current. Она возвращает результат в единицах LW. Это число мы считаем
за единицу рисунка, переводим в мм на бумаге, а потом - обратно в единицы
для ширины полилиний. Не забываем целое число перевести в действительное
-------------------------------------------------------------------|;
```

```
  (setq pline_width
    (ru-conv-millimeter-in-paper-to-unit
     (ru-conv-unit-to-millimeter-in-paper (* 1.0 (ru-lw-current))))
;|-------------------------------------------------------------------
Вычисляем требуемое имя блока, включающее параметры ширины, единиц и масштаба
-------------------------------------------------------------------|;
    name (ru-block-lw-name name pline_width scale))
  (if (tblsearch "BLOCK" name)
;|-------------------------------------------------------------------
Если блок с новым именем уже определен, стираем вставку прототипа
-------------------------------------------------------------------|;
   (progn (entdel ent) (setq do_insert t))
;|-------------------------------------------------------------------
А иначе начинаем создавать новый блок из прототипа. Прототип расчленяем, и сразу
удаляем, т. к. vla-explode после расчленения не удаляет основной объект.
Но зато расчленяет и разномасштабные вставки блоков.
Точку def_pnt в начале координат вводим исключительно из-за "глюка"
системы AutoCAD 2004 - помещение примитивов, образовавшихся при расчленении
не в то место, где находился расчленяемый блок
-------------------------------------------------------------------|;
(if
  (ru-block-obj-make-def-from-insert ent name (- ang) "0" acbyblock
       aclnwtbylwdefault pline_width)
      (setq do_insert t)
   )
)
;|-------------------------------------------------------------------
И вставляем новый блок на место прототипа
-------------------------------------------------------------------|;
  (if do_insert
    (ru-block-insert-attedit name ins_pnt scale scale scale ang)
 )
  do_insert)
```

Функция `ru-block-lw-name` (листинг 22.39) формирует имя, в которое входит базовое имя блока, ширина полилинии и масштаб. Ей может передаваться имя блока, в котором уже есть ширина — при редактировании существующих вставок.

Пример:

Исходное имя блока К-50-80-160, ширина полилинии 50 единиц, масштаб — 1. Будет возвращено имя блока RU-LW(К-50-80-160)50000-мм-1000.

Листинг 22.39. Функция `ru-block-lw-name`

```
(defun ru-block-lw-name (name pline_width scale / base)
 (strcat "RU-LW("
  (if (setq base (ru-string-word-or-nil name 2 (list "(" ")")))
       base name) ")" (itoa (fix (* pline_width 1000))) "-"
         (ru-unit-name) "-" (itoa (fix (* scale 1000)))
))
```

Создание определения нового блока из блока-прототипа производится функцией `ru-block-obj-make-def-from-insert` (листинг 22.40). Помимо создания блока имеется возможность изменить слой, цвет, вес линий и ширину полилиний у примитивов. Предусмотрен и обход ошибки системы AutoCAD 2004, связанной с расчленением блоков.

Листинг 22.40. Функция `ru-block-obj-make-def-from-insert`

```
(defun ru-block-obj-make-def-from-insert (insert_ent name new_ang new_layer
new_color new_lw new_width / insertion_point obj vla_array_objs)
 (setq obj (vlax-ename->vla-object insert_ent)
  insertion_point (vlax-safearray->list
                   (vlax-variant-value (vla-get-insertionpoint obj)))
  vla_array_objs (ru-obj-ent-ss-explode obj))
 (ru-obj-vla-array-move vla_array_objs insertion_point (list 0 0 0))
 (ru-obj-vla-array-rotate vla_array_objs (list 0 0 0) new_ang)
 (if new_layer (ru-obj-vla-array-mod vla_array_objs "Layer" new_layer))
 (if new_color (ru-obj-vla-array-mod vla_array_objs "Color" new_color))
 (if new_lw (ru-obj-vla-array-mod vla_array_objs "Lineweight" new_lw))
 (if new_width (ru-obj-vla-array-mod vla_array_objs "ConstantWidth" new_width))
 (ru-block-obj-make-from-vla-array name (list 0 0 0) vla_array_objs)
)
```

Здесь мы впервые столкнулись с программным созданием определения блока и с функциями-заменителями команд редактирования. Начнем с определения блока (листинг 22.41). Создание блока выполняется просто — функцией `vla-add`. При этом в семейство блоков добавляется пустой блок с заданным именем и базовой точкой. Его уже можно вставлять, но проку от этого мало — наоборот, один вред, т. к. пустой блок не удаляется командой PURGE (ОЧИСТИТЬ). А добавлять в определение блока примитивы приходится обходным путем. У объекта блока имеется много `Add`-методов для всех типов примитивов, но нет добавления множества объектов. Пришлось конвертировать набор примитивов в безопасный массив и использовать метод `vla-copyobjects`.

Листинг 22.41. Функция `ru-block-obj-make-from-vla-array`

```
(defun ru-block-obj-make-from-vla-array (name base_point arr_objects / obj)
;;; Возвращает VLA-объект определения блока
 (ru-error-catch (function (lambda ()
  (vla-copyobjects (ru-obj-get-active-document) arr_objects
    (setq obj
        (vla-add (ru-obj-get-blocks) (vlax-3d-point base_point) name)))
        (ru-obj-vla-array-erase arr_objects)
        obj))
  (function (lambda (msg) (princ
    (strcat "\nRU-BLOCK-OBJ-MAKE-FROM-VLA-ARRAY: " msg)) nil))
);_ end of ru-error-catch
)
```

Для операций с массивом VLA-объектов используются функции, приведенные в следующих четырех листингах (листинг 22.42—22.45).

Листинг 22.42. Функция ru-obj-vla-array-rotate

```
(defun ru-obj-vla-array-rotate (vla_array_objs base_point rad_angle)
  (foreach obj
    (vlax-safearray->list (vlax-variant-value vla_array_objs))
     (ru-obj-ent-ss-rotate obj base_point rad_angle)))
```

Листинг 22.43. Функция ru-obj-vla-array-scale

```
(defun ru-obj-vla-array-scale (vla_array_objs base_point scale)
  (foreach obj
    (vlax-safearray->list (vlax-variant-value vla_array_objs))
    (ru-obj-ent-ss-scale obj base_point scale)))
```

Листинг 22.44. Функция ru-obj-vla-array-move

```
(defun ru-obj-vla-array-move (vla_array_objs pnt_from pnt_to)
  (foreach obj
     (vlax-safearray->list (vlax-variant-value vla_array_objs))
     (ru-obj-ent-ss-move obj pnt_from pnt_to)))
```

Листинг 22.45. Функция ru-obj-vla-array-erase

```
(defun ru-obj-vla-array-erase (vla_array_objs)
  (foreach obj
    (vlax-safearray->list (vlax-variant-value vla_array_objs))
    (ru-obj-ent-ss-erase obj)))
```

Изменение значения свойства объекта производится с помощью функции, приведенной в листинге 22.46, а изменение значения свойства у массива объектов — с помощью функции, приведенной в листинге 22.47.

Листинг 22.46. Функция ru-obj-modify-prop

```
(defun ru-obj-modify-prop (obj prop_name value)
  (cond ((vlax-property-available-p obj prop_name T)
         (vlax-put-property obj prop_name
           (vlax-make-variant
             (cond ((= (strcase value) "BYLAYER") acbylayer)
                   ((= (strcase value) "BYBLOCK") acbyblock)
                   (T value))))
         T
       )
        (T nil)
 );_ end of cond
)
```

Листинг 22.47. Функция `ru-obj-vla-array-mod`

```
(defun ru-obj-vla-array-mod (vla_array_objs prop_name value)
  (foreach obj
    (vlax-safearray->list (vlax-variant-value vla_array_objs))
    (ru-obj-modify-prop obj prop_name value)))
```

Теперь рассмотрим несколько функций, предназначенных для редактирования (листинги 22.48—22.53). По аналогичной схеме разработаны и другие подобные функции (`ru-obj-ent-ss-copy`, `ru-obj-ent-ss-mirror`). Всем им передается в качестве первого аргумента VLA-объект, примитив или набор. Остальные аргументы соответствуют назначению функции. Возвращается `T` или `nil`.

Листинг 22.48. Функция `ru-obj-ent-ss-rotate`

```
(defun ru-obj-ent-ss-rotate (some base_point rad_angle / obj i)
  (ru-error-catch (function (lambda ()
        (setq base_point (vlax-3d-point base_point))
        (cond
          ((= (type some) 'vla-object)
           (vla-rotate some base_point rad_angle))
          ((= (type some) 'ename)
           (setq obj (vlax-ename->vla-object some))
           (vla-rotate obj base_point rad_angle)
           (vlax-release-object obj))
          ((= (type some) 'pickset)
           (setq i 0)
           (repeat (sslength some)
             (setq obj (vlax-ename->vla-object (ssname some i)) i(1+ i))
             (vla-rotate obj base_point rad_angle)
             (vlax-release-object obj)))
       );_ end of cond
        T))
    (function (lambda (msg)
     (princ (strcat "\nRU-OBJ-ENT-SS-ROTATE: " msg)) nil)))
)
```

Листинг 22.49. Функция `ru-obj-ent-ss-scale`

```
(defun ru-obj-ent-ss-scale (some base_point scale / i obj)
  (ru-error-catch (function (lambda ()
     (setq base_point (vlax-3d-point base_point))
     (cond
       ((= 'PICKSET (type some))
        (setq i 0)
        (repeat (sslength some)
          (setq obj (vlax-ename->vla-object (ssname some i)) i (1+ i))
          (vla-scaleentity obj base_point scale)
         (vlax-release-object obj)))
```

```
      ((= 'VLA-OBJECT (type some))
        (vla-scaleentity some base_point scale))
      ((= 'ENAME (type some))
        (setq obj (vlax-ename->vla-object some))
        (vla-scaleentity some base_point scale)
        (vlax-release-object obj))
      ) T))
   (function (lambda (msg)
       (princ (strcat "\nRU-OBJ-ENT-SS-SCALE: " msg)) nil)))
)
```

Листинг 22.50. Функция `ru-obj-ent-ss-erase`

```
(defun ru-obj-ent-ss-erase (some / i)
  (ru-error-catch (function (lambda ()
    (cond
      ((= 'PICKSET (type some))
       (setq i 0)
       (repeat (sslength some)
         (vla-delete (vlax-ename->vla-object (ssname some i)))
         (setq i (1+ i))))
      ((= 'VLA-OBJECT (type some)) (vla-delete some))
      ((= 'ENAME (type some)) (vla-delete (vlax-ename->vla-object some)))
   ) T))
    (function (lambda (msg)
      (princ (strcat "\nRU-OBJ-ENT-SS-ERASE: " msg)) nil)))
)
```

Листинг 22.51. Функция `ru-obj-ent-ss-move`

```
(defun ru-obj-ent-ss-move (some pnt_from pnt_to / i obj)
  (ru-error-catch (function (lambda ()
     (setq pnt_from (vlax-3d-point pnt_from)
           pnt_to (vlax-3d-point pnt_to))
     (cond
      ((= 'PICKSET (type some))
        (setq i 0)
        (repeat (sslength some)
          (setq obj (vlax-ename->vla-object (ssname some i)) i (1+ i))
          (vla-move obj pnt_from pnt_to) (vlax-release-object obj)
        )
      )
       ((= 'VLA-OBJECT (type some)) (vla-move some pnt_from pnt_to))
       ((= 'ENAME (type some))
        (setq obj (vlax-ename->vla-object obj))
        (vla-move obj pnt_from pnt_to) (vlax-release-object obj)
      )
    ) T))
   (function (lambda (msg)
     (princ (strcat "\nRU-OBJ-ENT-SS-MOVE: " msg)) nil)))
)
```

Особенно полезной оказалась функция расчленения `ru-obj-ent-ss-explode` (листинг 22.52). Помимо обработки блоков, ее мы использовали для создания функции-команды C:RU−EXPLODE (листинг 22.53), расчленяющей разномасштабные вставки блока и помещающей образовавшиеся примитивы на текущий слой.

Замечание

Расчленение разномасштабных вставок блоков с использованием метода Explode прекрасно работало в системе AutoCAD 2002 и перестало выполняться в AutoCAD 2004 и AutoCAD 2005. Очередная неприятность со стороны фирмы Autodesk, иначе не назовешь такие сюрпризы, приводящие к неработоспособности большой цепочки функций сторонних разработчиков. Способность команды EXPLODE (РАСЧЛЕНИТЬ) расчленять блоки с разномасштабными вставками управляется значением системной переменной EXPLMODE, но на объектную модель эта переменная не действует. Приехали.

Дополнительное исследование показало, что в AutoCAD 2004 и AutoCAD 2005 расчленение не получается в тех случаях, когда масштаб по оси Z отличается от масштаба по оси X. В результате пришлось в функцию `ru-obj-ent-ss-explode` вставлять проверку типа объекта и приведение масштаба по оси Z к масштабу по оси X. Этого недостаточно для гарантированного расчленения блоков, но временно придется смириться, а во всех наших функциях вставки блока масштаб по оси Z предусмотреть таким же, как по оси X.

Листинг 22.52. Функция `ru-obj-ent-ss-explode`

```
(defun ru-obj-ent-ss-explode (some / obj i result last_ent count)
 (defun _explode (obj / result)
  (if (vlax-method-applicable-p obj 'explode)
   (progn
    (if (= (vla-get-objectname obj) "AcDbBlockReference")
     (vla-put-zscalefactor obj (vla-get-xscalefactor obj)))
    (setq result (vla-explode obj))
    (vla-delete obj)))
  result
)
 (ru-error-catch (function (lambda ()
   (cond
     ((= 'pickset (type some))
      (setq i 0 count 0 last_ent (entlast))
      (repeat (sslength some)
       (setq obj (vlax-ename->vla-object (ssname some i)) i (1+ i))
       (if (_explode obj) (setq count (1+ count)))
     )
      (if (> count 0)
       (setq result
        (ru-ss-to-vla-array (ru-ss-select-after-ent last_ent)))))
     ((= 'vla-object (type some))
      (setq result (_explode some))
    )
     ((= 'ename (type some))
      (setq result (_explode (vlax-ename->vla-object some))))
   )
```

```
    result
  ))
  (function (lambda (msg)
         (princ (strcat "\nRU-OBJ-ENT-SS-EXPLODE: " msg)) nil))
)
)
```

Листинг 22.53. Функция C:RU-EXPLODE

```
(defun c:ru-explode (/ vla_array_objs)
(if (setq vla_array_objs (ru-obj-ent-ss-explode (ru-ss-get)))
  (progn
   (foreach obj (vlax-safearray->list vla_array_objs)
    (ru-obj-modify-prop obj "Layer" (getvar "CLAYER"))
    (ru-obj-modify-prop obj "Color" (getvar "CECOLOR"))
  )))
 (princ)
)
```

Однако, мы отвлеклись от вставки блоков. Вставка блока с изменяемой шириной из библиотеки будет выполняться функцией, приведенной в листинге 22.54.

Листинг 22.54. Функция ru-block-lw-lib-insert-ptask-angleask

```
(defun ru-block-lw-lib-insert-ptask-angleask (block_lib  block_name
   scale)
 (ru-app-begin)
  (if (ru-block-lib-insert block_lib block_name)
    (ru-block-lw-insert-ptask-angleask block_name scale)
 )
 (ru-app-end)
 (princ)
)
```

Изменение ширины линий во вставленном блоке возможно с помощью функции, приведенной в листинге 22.55.

Листинг 22.55. Функция ru-block-lw-edit

```
(defun ru-block-lw-edit (/edata ent)
  (if (setq ent (ru-get-entsel-by-type
   "Укажи блок для изменения ширины линий" "Это не INSERT"
     (list "INSERT") T))
    (progn
      (setq edata (entget (setq ent (car ent))))
      (ru-dlg-dcl-select-lw)
      (ru-block-lw-change ent (cdr (assoc 2 edata))
        (cdr (assoc 10 edata)) (cdr (assoc 50 edata))
        (cdr (assoc 41 edata)))))
 (princ)
)
```

Мы привели примеры функций интерактивной вставки отдельных блоков с изменяемой шириной линий, но их можно использовать и в более сложных функциях, комбинируя с рисованием других объектов. Хитроумно сконструированные блоки такого типа позволяют перевести работу по расширению нашей системы в ранг адаптации, доступной обычным пользователям.

Врезка блоков и текстов в линии

Очень большое количество задач решается путем врезки (или прикрепления) объекта в линии. Чаще всего этот прием применяется в сантехнических и электрических схемах, но может использоваться и в других разделах. Схему, насыщенную различными условными изображениями, можно рисовать так: нарисовать линейный участок, вставить в конце блок (помучившись с прицеливанием и поворотом), от конца блока продолжить рисование линии. Можно и "прикреплять" блоки к нарисованной линии, а потом вырезать кусочки, занятые блоками. А можно и сразу — нарисовать схему в линиях, а потом одним указанием точки на линии вставлять выбранный блок или текст, который сам будет разворачиваться и вырезать, при необходимости, участок линии.

Для реализации задуманного потребуется решение нескольких задач. Начнем с указания точки на линии (листинг 22.56). Очень важно, как мы устанавливаем размеры прицела, применяем объектную привязку к ближайшей точке при указании и еще раз к ближайшей точке при возврате результата. Это позволяет избежать ошибок при указании точки на *широкой* полилинии. Возвращается результат в виде списка из имени указанного примитива и точки указания в *мировой системе координат* (МСК).

Листинг 22.56. Функция `ru-get-point-on-ent`

```
(defun ru-get-point-on-ent (msg / ent_selected old_aperture om result)
;|
 Выбор точки на примитиве. Возвращает список из имени примитива и точки указания
Пример:
 (ru-get-point-on-ent "Укажи точку на отрезке или полилинии")
 (<Entity name: 4008f2b0> (898.383 270.931 0.0))
Точка возвращается в МСК
|;
 (setq om (getvar "OSMODE") old_aperture (getvar "APERTURE"))
 (setvar "OSMODE" 512) (setvar "APERTURE" (getvar "PICKBOX"))
 (if (setq ent_selected
    (ru-get-entsel-by-type msg "Это не ОТРЕЗОК и не ПОЛИЛИНИЯ"
      (list "LINE" "LWPOLYLINE") T))
      (setq result (list (car ent_selected)
        (trans (osnap (trans (cadr ent_selected) 0 1) "_nea") 1 0)))
)
(setvar "OSMODE" om) (setvar "APERTURE" old_aperture)
 result
)
```

Далее необходимо получить координаты указанного сегмента (листинги 22.57—22.58) линейного объекта.

Листинг 22.57. Функция `ru-geom-list-ent-point`

```
(defun ru-geom-list-ent-point (ent pick_point / lst_points result)
(if (setq lst_points
  (ru-geom-segm-line-by-point pick_point (ru-pline-list-vertex ent)))
    (setq result (list (car lst_points) (cadr lst_points)))
    (princ (strcat
        "Не найден сегмент, пригодный для врезки!"
        "\nВозможно, указана дуга или захвачен посторонний объект!")))
  result
)
```

Листинг 22.58. Функция `ru-geom-segm-line-by-point`

```
(defun ru-geom-segm-line-by-point (test_point lst_points / is_closed  pt1 pt2
n result)
 (setq is_closed (cadr lst_points) lst_points (car lst_points) n 0)
;; добавляем фиктивную замыкающую вершину
 (if is_closed (setq lst_points (cons (last lst_points) lst_points)))
 (while (and (not result) (< n (- (length lst_points) 1)))
     (setq pt1 (nth n lst_points) pt2 (nth (1+ n) lst_points))
     (if (ru-geom-is-point-in-line pt1 pt2 test_point)
       (setq result  (list pt1 pt2))
       (setq n (1+ n))
    )
)
 (if (and is_closed result) (setq result (reverse result)))
  result
)
```

Теперь нам известны примитив, точка указания и координаты начала и конца сегмента, на котором указана точка. Этих данных достаточно, чтобы выполнять необходимые преобразования и создание объектов. Некоторые варианты мы еще разберем в последующих главах, а сейчас закончим с врезкой блоков. Теперь нам необходимо вставить в указанную точку блок и факультативно вырезать кусочек сегмента (иногда, например, для изображений футляров, стояков, стыков, вырезать не нужно). Функция `ru-pline-break-length` (листинг 22.59) вырезает по приглашению `msg` в отрезке или полилинии участок длиной `break_length` (если `break_length` = 0, то участок не вырезается) и возвращает список из точки центра разрыва и угла наклона отрезка или `nil`. Мы будем использовать старую добрую команду BREAK (РАЗОРВАТЬ), т. к. реализовать разрыв линий объектными методами очень сложно, хотя и возможно.

Листинг 22.59. Функция `ru-pline-break-length`

```
(defun ru-pline-break-length (msg break_length / angle_segm can_break ent
     ent_point lst_segm pnt1_found pnt1_sel_segm pnt2_found pnt2_sel_segm
     pnt_selected result)
```

```
;|
Примеры:
 (ru-pline-break-length "Выбери линию для врезки" 100)
 (ru-pline-break-length "Выбери линию для врезки" 0)
|;
 (if (setq ent_point (ru-get-point-on-ent msg))
  (if (setq lst_segm (ru-geom-list-ent-point (setq ent (car ent_point))
                       (setq pnt_selected (cadr ent_point))))
    (setq pnt1_sel_segm (car lst_segm) pnt2_sel_segm (cadr lst_segm)
          can_break T)))
  (if can_break
   (progn
     (setq angle_segm (angle pnt1_sel_segm pnt2_sel_segm))
     (if (> break_length 0)
       (progn
         (setq break_length (/ break_length 2))
;|-------------------------------------------------------------------------
Находим точки вырезки и проверяем, вписываемся ли в габариты сегмента
-------------------------------------------------------------------------|;
         (if (>= (distance pnt_selected pnt1_sel_segm) break_length)
            (setq pnt1_found T
                  pnt1_sel_segm
                  (polar pnt_selected angle_segm break_length))
             (setq pnt1_found nil
                   pnt1_sel_segm
                   (polar pnt_selected angle_segm (* break_length 2)))
         )
          (if
            (>= (distance pnt_selected pnt2_sel_segm) break_length)
             (setq pnt2_found T
                   pnt2_sel_segm
                   (polar pnt_selected (ru-geom-go-back angle_segm)
                        break_length))
             (setq pnt2_found nil
                   pnt2_sel_segm
                   (polar pnt_selected (ru-geom-go-back angle_segm)
                         (* break_length 2)))
         )
          (ru-var-clear-osnap)
;|-----------------------------------------------------------------
Для команды передаем точки в ПСК
-----------------------------------------------------------------|;
          (cond
            (pnt1_found
             (command "_.BREAK" (list ent (trans pnt_selected 0 1))
             "_F" (trans pnt1_sel_segm 0 1) (trans pnt2_sel_segm 0 1))
             (setq result (list pnt_selected angle_segm))
           )
            (pnt2_found
             (command "_.BREAK" (list ent (trans pnt_selected 0 1))
```

```
            "_F" (trans pnt2_sel_segm 0 1) (trans pnt1_sel_segm 0 1))
            (setq result (list pnt_selected angle_segm))
          )
           (T (ru-msg-alert "\nНе вписываемся в указанный сегмент"))
        )
         (ru-var-restore-osnap)
      );_ end of progn
       (setq result (list pnt_selected angle_segm))
    );_ end of if
   );_ end of progn
 );_ end of if
  result
)
```

Осталось разработать функции вставки блоков. Функции `ru-block-insert-align` (листинг 22.60) в аргументах передается имя библиотеки блоков, имя блока, масштабы вставки по X и Y. Аргумент `is_ask_rotate` указывает на необходимость запроса на переворот блока (изображение может быть несимметричным), аргумент `is_break_line` — на необходимость вырезки линии.

Листинг 22.60. Функция `ru-block-insert-align`

```
(defun ru-block-insert-align (block_lib_name block_name x y is_ask_rotate
is_break_line / lst_break)
 (if (ru-block-lib-insert block_lib_name block_name)
  (while
   (setq lst_break (ru-pline-break-length "Точка вставки на линии"
          (if is_break_line (ru-conv-millimeter-in-paper-to-unit x) 0)))
   (ru-block-insert-obj block_name (car lst_break)
           (ru-conv-millimeter-in-paper-to-unit x)
           (ru-conv-millimeter-in-paper-to-unit y) 1
           (ru-conv-rad-to-deg (cadr lst_break)))
   (if (and is_ask_rotate (not (ru-dlg-no-cml "Перевернуть")))
      (ru-obj-ent-ss-rotate (entlast) (car lst_break) pi))
  ))
  lst_break
)
```

Конечные программы для включения в меню будут сводиться к нескольким вариантам вызова этой функции. Их мы рассмотрим в *части IV* книги.

Аналогичным образом разработаем функцию для врезки текстов. Текст может врезаться в линию или быть написан вдоль линии (вариант с текстовым типом линии рассмотрим позже). Вначале разработаем универсальную функцию привязки текста к линии (листинг 22.61).

Листинг 22.61. Функция `ru-draw-txt-up-or-in-line`

```
(defun ru-draw-txt-up-or-in-line (msg txt  break_len / break_lst text_pnt angle_rad
angle_txt txt_height)
```

```
 (while
  (setq break_lst
   (ru-pline-break-length (strcat msg " '" TXT "'") break_len))
  (setq text_pnt (car break_lst) txt_height (ru-normal-text-height)
             angle_rad  (cadr break_lst))
  (cond
    ((and (> angle_rad 1.5708) (<= angle_rad 4.71239))
      (setq angle_txt (ru-geom-go-back angle_rad))
      (if (= break_len 0)
       (setq text_pnt
         (polar text_pnt (ru-geom-go-right angle_rad) txt_height)))
   )
    (t (setq angle_txt angle_rad)
      (if (= break_len 0)
       (setq text_pnt
         (polar text_pnt (ru-geom-go-left angle_rad) txt_height)))
   )
 );_ end of cond
  (ru-text-add txt text_pnt txt_height angle_txt
     (if (= break_len 0) acAlignmentCenter acAlignmentMiddle)
 )
)
```

Теперь функция для рисования врезанного текста (листинг 22.62) будет очень проста, а для рисования текста вдоль линии (листинг 22.63) — еще проще.

Листинг 22.62. Функция `ru-draw-txt-in-line`

```
(defun ru-draw-txt-in-line (txt / lst)
  (if (/= txt "")
   (progn
      (setq lst (textbox (list (cons 1 txt))))
      (ru-draw-txt-up-or-in-line "Точка на линии для врезки текста"
        txt (+ (- (car (cadr lst)) (car (car lst)))
           (ru-conv-millimeter-in-paper-to-unit 2)))
   ))
)
```

Листинг 22.63. Функция `ru-draw-txt-up-line`

```
(defun ru-draw-txt-up-line (txt)
  (ru-draw-txt-up-or-in-line "Точка на линии для центра текста" txt 0)
)
```

Пример изображения, созданного с использованием функций врезки блоков и текстов, показан на рис. 22.3.

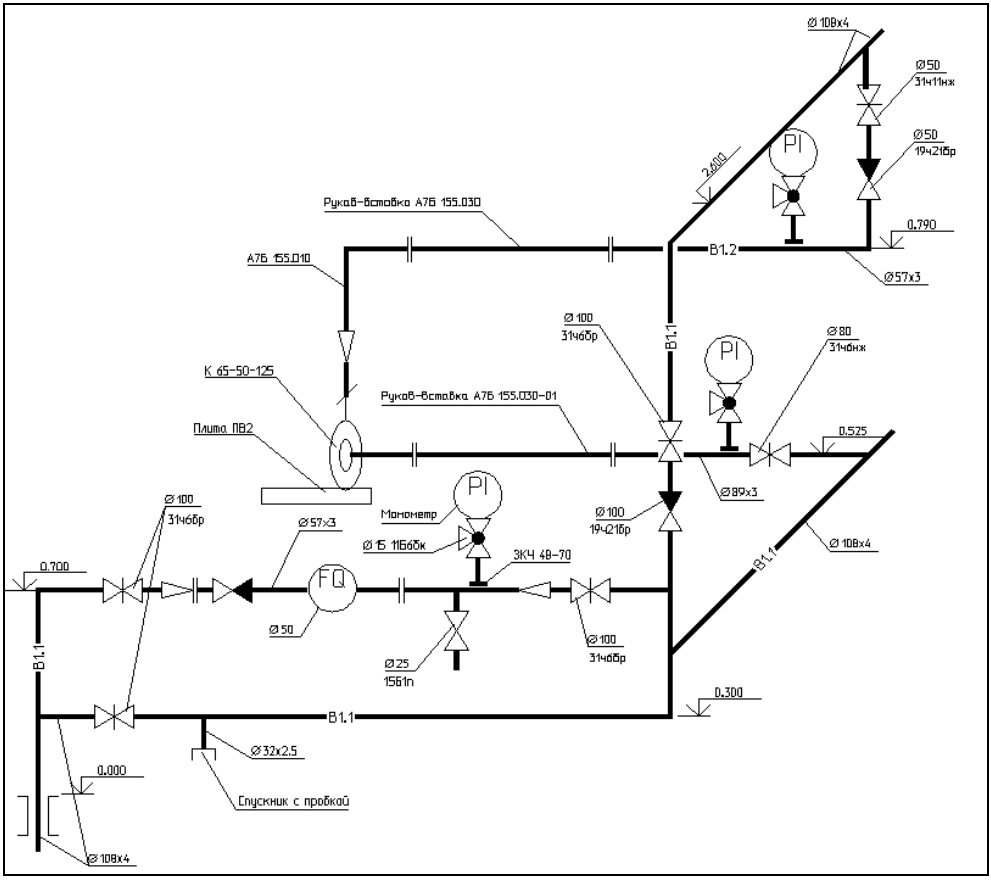


Рис. 22.3. Пример врезки блоков и текстов в линии

Рисование объектов

Воспроизводить функции рисования, основанные на использовании команд, мы не будем — это слишком просто, а основные принципы мы рассмотрели в *главе 10*. Остановимся на рисовании объектными методами, иногда, для сравнения, рассматривания методику с `entmake`. Напоминаем, что координаты точек всегда передаются в МСК.

Создание отрезков

Начнем с рисования отрезков. Функция, приведенная в листинге 22.64, создает отрезок с заданными координатами, весом и типом линии. При создании примитивов объектными методами мы будем придерживаться следующих принципов:

1. Сначала создается объект основным ADD-методом (в данном случае `AddLine`).
2. Устанавливаются требуемые свойства объекта.
3. Для надежности производятся проверки возможности изменения объектов.
4. Все окружается ловушкой ошибок, позволяющей, в случае сбоя по непредсказуемым причинам, корректно завершить функцию с выводом сообщения и возвратом `nil`.

Листинг 22.64. Функция `ru-line-add`

```
(defun ru-line-add (start_pnt end_pnt lineweight ltype / obj)
  (ru-error-catch (function (lambda ()
     (setq obj (vla-addline (ru-obj-active-space)
           (vlax-3d-point start_pnt) (vlax-3d-point end_pnt)))
     (cond
       ((vlax-write-enabled-p obj)
         (if lineweight (vla-put-lineweight obj lineweight))
         (if Ltype (ru-ltype-apply obj Ltype))
         (vla-update obj)
       ))))
   (function (lambda (x) (princ (strcat "\nОШИБКА RU-LINE-ADD " x))
               nil))))
```

Создать сразу несколько отрезков по заданному списку координат можно функцией, приведенной в листинге 22.65.

Листинг 22.65. Функция `ru-line-add-multi`

```
(defun ru-line-add-multi (lst_points is_closed lineweight ltype / pt1 pt_last)
 (setq pt1 (car lst_points) pt_last (last lst_points))
  (while (and lst_points (>= (length lst_points) 2))
    (ru-line-add (car lst_points) (cadr lst_points) lineweight ltype)
    (setq lst_points (cdr lst_points))
 )
  (if is_closed (ru-line-add pt1 pt_last lineweight ltype))
  pt_last
)
```

Создание полилиний

В своей системе мы используем компактные (легкие) полилинии (по крайней мере в рассматриваемом круге задач). Создание полилинии объектным методом показано в листинге 22.66.

Листинг 22.66. Функция `ru-pline-add`

```
(defun ru-pline-add (points is_closed width lineweight ltype / obj)
 (ru-error-catch (function (lambda ()
```

```
  (setq obj (vla-addlightweightpolyline (ru-obj-active-space)
   (ru-conv-list-points-to-variant-array (apply 'append
                  (mapcar 'ru-conv-3dpoint-to-2dpoint points)))))
    (cond ((and (vlax-read-enabled-p obj) (vlax-write-enabled-p obj))
           (if is_closed (vla-put-closed obj :vlax-true))
           (if width (vla-put-constantwidth obj width))
           (if lineweight (vla-put-lineweight obj lineweight))
           (if Ltype (ru-ltype-apply obj Ltype))
           (vla-update obj)
           obj))))
    (function (lambda (x) (princ (strcat "\nОШИБКА RU-PLINE-ADD " x))
                nil)))
)
```

Для сравнения приводим функцию создания полилинии с помощью функции `entmake` (листинг 22.67). В этой функции мы позволяем создавать и трехмерную полилинию (POLYLINE).

Листинг 22.67. Функция `ru-pline-entmake`

```
(defun ru-pline-entmake (points is_closed is_3d width lineweight / elst entl)
  (if is_3d
    (progn
      (setq entl
       (list '(0 . "POLYLINE") '(100 . "AcDbEntity")
       '(100 . "AcDb3dPolyline") '(66 . 1) '(10 0.0 0.0 0.0)
       (cons 70 (logior 8 (if is_closed 1 0)
         (if (= 1 (getvar "PLINEGEN")) 128 0))))
     );_ end of setq
      (if (entmake entl)
        (progn
          (foreach v points
              (setq entl (list '(0 . "VERTEX") '(100 . "AcDbEntity")
                '(100 . "AcDbVertex") '(100 . "AcDb3dPolylineVertex")
                (append '(10) v) '(70 . 32))
              )
               (entmake entl)
         );_ end of foreach
          (if (entmake '((0 . "SEQEND") (100 . "AcDbEntity")))
            (entupd (entlast))
           nil)))
   );_ end of progn
    (progn
      (setq elst
        (append
         (list '(0 . "LWPOLYLINE") '(100 . "AcDbEntity")
           '(100 . "AcDbPolyline") (cons 90 (length points))
           (cons 43 width) (cons 370 lineweight)
           (cons 70 (logior (if is_closed 1 0)
```

```
            (if (= 1 (getvar "PLINEGEN")) 128 0)))
        )
         (mapcar (function (lambda (p) (list 10 (car p) (cadr p))))
                           points)
       );_ end of append
     );_ end of setq
     (if (entmake elst) (entupd (entlast)) nil)
   );_ end of progn
 );_ end of if
)
```

Создание текстов

Как создавать текстовые примитивы с помощью функции `command` мы разбирали в *главе 10*. Теперь разработаем функцию создания текстов объектными методами (листинг 22.68). Аргументами функции являются текстовая строка, точка вставки текста, высота, поворот и выравнивание. Прежде выравнивание мы обозначали символами (L, M, C и т. д.), теперь же решили обозначать такими же целыми числами, как и в AutoCAD, а для облегчения восприятия числа заменять глобальными константами AutoCAD:

- 0 — `acAlignmentLeft`;
- 1 — `acAlignmentCenter`;
- 2 — `acAlignmentRight`;
- 3 — `acAlignmentAligned`;
- 4 — `acAlignmentMiddle`;
- 5 — `acAlignmentFit`;
- 6 — `acAlignmentTopLeft`;
- 7 — `acAlignmentTopCenter`;
- 8 — `acAlignmentTopRight`;
- 9 — `acAlignmentMiddleLeft`;
- 10 — `acAlignmentMiddleCenter`;
- 11 — `acAlignmentMiddleRight`;
- 12 — `acAlignmentBottomLeft`;
- 13 — `acAlignmentBottomCenter`;
- 14 — `acAlignmentBottomRight`.

Использование именованных констант позволяет хорошо понимать текст и исключить значительные участки кода, преобразующие текстовые обозначения в целочисленные. При использовании выравниваний `acAlignmentAligned` и `acAlignmentFit` вместо точки вставки текста должен передаваться список из двух точек — первой и второй точек для выравнивания или вписывания текстовой строки.

Листинг 22.68. Функция `ru-text-add`

```
(defun ru-text-add (txt pnt height rotation justification / obj)
(if (null justification) (setq justification acalignmentleft))
  (ru-error-catch (function (lambda ()
   (setq obj (vla-addtext (ru-obj-active-space) txt
     (if (or (= justification acalignmentaligned)
             (= justification acalignmentfit))
               (vlax-3d-point (car pnt))
               (vlax-3d-point pnt)
    ) height))
```

```
   (vla-put-lineweight obj (ru-lw-calc-for-text height))
    (cond
      ((= justification acalignmentleft) (vla-put-rotation obj rotation))
      ((or (= justification acalignmentaligned)
           (= justification acalignmentfit))
       (vla-put-alignment obj justification)
       (vla-put-textalignmentpoint obj (vlax-3d-point (cadr pnt)))
     )
     (T
       (vla-put-alignment obj justification)
       (vla-put-textalignmentpoint obj (vlax-3d-point pnt))
       (vla-put-rotation obj rotation)
     )
   );_ end of cond
    (vla-update obj) obj))
    (function (lambda (x)
       (princ (strcat "\nОШИБКА RU-TEXT-ADD: " x)) nil))
 )
)
```

При создании текста мы установили для него вес линий с помощью функции `ru-lw-calc-for-text height` (листинг 22.69). Сделали мы это из самых добрых побуждений — часто пользователи жалуются, что тексты слишком "тонкие". Применять TTF-шрифты ради "жирности" текста не хочется. Мы устанавливаем вес линий текста 5% от высоты, хотя по стандарту требуется 7—10%. Результат, прямо скажем, не очень хороший — при включенном режиме **LWT** тексты становятся нечитаемыми и правильно выглядят только тогда, когда масштаб изображения на экране примерно соответствует масштабу на бумаге. На бумаге-то все может и будет хорошо, но излишняя "назойливость" отображения веса линий — большой недостаток системы AutoCAD, и не только в отношении текстов.

Листинг 22.69. Функция `ru-lw-calc-for-text` — вариант 1

```
(defun ru-lw-calc-for-text (text_height)
 (ru-conv-millimeter-to-acadlw
  (ru-conv-unit-to-millimeter (* 0.05 text_height))))
```

В строительном проектировании подавляющее количество надписей выполняется с минимальной высотой текста (по нашему "стандарту" — 2,5 мм на бумаге), а редкие крупные тексты особенно хочется видеть "жирными". Изменим функцию так, чтобы вес линий устанавливался дифференцированно (листинг 22.70).

Листинг 22.70. Функция `ru-lw-calc-for-text` — вариант 2

```
(defun ru-lw-calc-for-text (text_height)
(if (> (ru-conv-unit-to-millimeter-in-paper text_height) 3.0)
 (ru-conv-millimeter-to-acadlw
   (ru-conv-unit-to-millimeter (* 0.05 text_height)))
    (getvar "LWDEFAULT")))
```

Изменение веса линий

Если уж мы упомянули про недостаток веса линий, то напишем простую функцию, позволяющую изменить вес полилиний на эквивалентную физическую ширину полилиний. Так же, как и для блоков с изменяемой шириной линий, в диалоге выбирается вес линий, но использоваться он будет для определения эквивалентной ширины полилиний. Изменить ширину полилиний очень просто (листинг 22.71).

Листинг 22.71. Функция `ru-lw-replace-pline-width`

```
(defun ru-lw-replace-pline-width (/ selection)
 (princ "\nВыбери полилинии для изменения ВЕСА на эквивалентную ШИРИНУ")
 (if (setq selection (ssget '((0 . "LWPOLYLINE"))))
 (progn
  (ru-dlg-dcl-select-lw)
  (ru-obj-vla-array-mod
    (vlax-make-variant
      (ru-ss-to-vla-array (ru-ss-remove-locked selection)))
       "ConstantWidth"
       (ru-conv-millimeter-in-paper-to-unit
         (ru-conv-unit-to-millimeter-in-paper (* 1.0 (ru-lw-current)))
       ))))
  (princ)
)
```

Чем эта функция лучше изменения ширины полилиний стандартными средствами, например, через окно свойств объектов? Диалог редактирования свойств хорош, но слишком универсален. Пользователь должен предварительно *вычислить*, какое именно значение "Global width" в *единицах рисунка* следует ввести в диалоговом окне, чтобы получить желаемый результат, а функция позволяет выбрать ширину не в единицах рисунка, а в *миллиметрах на бумаге*, причем из нашего стандартного ряда.

Создание кругов

Рисовать круги приходится довольно часто, делать это мы будем специальной функцией (листинг 22.72), не требующей пояснений.

Листинг 22.72. Функция `ru-circle-add`

```
(defun ru-circle-add (center_pnt radius lineweight ltype / obj)
  (ru-error-catch (function(lambda ()
    (setq obj (vla-addcircle (ru-obj-active-space)
           (vlax-3d-point center_pnt) Radius))
   (cond
     ((vlax-write-enabled-p obj)
      (if lineweight (vla-put-lineweight obj lineweight))
      (if Ltype (ru-ltype-apply obj Ltype))
      (vla-update obj) obj))))
```

```
   (function
   (lambda (x) (princ (strcat "\nОШИБКА RU-CIRCLE-ADD: " x)) nil))
 )
)
```

Семейство функций для рисования трасс и линий

По частоте применения после блоков и текстов идет рисование различных линейных объектов. Часто можно использовать и стандартные команды, но для специализированных изображений потребуется рисование универсальных трасс и линий, легко модифицируемых в узкоспециальные. Примером такой функции может служить функция `ru-trass-draw-first-prev-sample` (листинг 22.73). Ей передается единственный аргумент — имя функции рисования сегмента. Функция рисования сегмента может делать что угодно при единственном условии — она получает в качестве аргументов точки начала и конца сегмента в мировой системе координат. При этом само рисование необязательно (хотя чаще всего что-то изображается) — может, например, просто производиться суммирование длин сегментов. Основная функция запрашивает в командной строке точки сегментов.

Первая точка [Prev/Sample/New]<Выход>:
Следующая точка [Prev/Sample/New]<Выход>:
Следующая точка [Prev/Sample/New]<Выход>:

При выборе опции **Prev** функция "цепляется" к начальной точке последнего сегмента и продолжает запросы.

Следующая точка [Prev/Sample/New]<Выход>:
Следующая точка [Prev/Sample/New]<Выход>:

При выборе опции **New** запрашивается новая начальная точка и продолжается работа.

Первая точка [Prev/Sample/New]<Выход>:
Следующая точка [Prev/Sample/New]<Выход>:

При выборе опции **Sample** запрашивается выбор отрезка или полилинии и по всем сегментам полилинии выполняется функция рисования, после чего функция цепляется за конечную точку последнего сегмента и продолжает запросы.

Выбери отрезок или полилинию <Выход>:
Следующая точка [Prev/Sample/New]<Выход>:

Листинг 22.73. Функция `ru-trass-draw-first-prev-sample`

```
(defun ru-trass-draw-first-prev-sample (quoted_func / first_point list_point
next_point ent list_vertex segment_points)
(while (setq next_point (_ru-get-with-default
 (if first_point "Следующая точка" "Первая точка")
          "Выход" 'getpoint nil
```

```
 (if (or first_point *ru_last_end_point*) "Prev Sample New" "Sample New")
 (if (ru-is-point first_point) (trans first_point 0 1) first_point)))
;; Возвращенная точка преобразуется в МСК
 (if (ru-is-point next_point)
   (setq next_point (trans next_point 1 0)) next_point)
 (cond
  ((= next_point "Sample")
   (if (setq ent (ru-get-entsel-by-type "Выбери отрезок или полилинию"
        "Это не ОТРЕЗОК и не ПОЛИЛИНИЯ" (list "LINE" "LWPOLYLINE") NIL))
    (progn
     (setq list_vertex (ru-pline-list-vertex (car ent))
     segment_points (ru-geom-get-segment-points (car list_vertex)
                                  (cadr list_vertex)))
     (foreach segment segment_points
             ((eval quoted_func) (setq first_point (car segment))
               (setq next_point (cadr segment)))
             (setq list_point  (cons first_point list_point)
                  list_point  (cons next_point list_point)
                  first_point next_point))
   );_ end of progn
  );_ end of if
 )
  ((= next_point "New") (setq first_point nil))
  ((= next_point "Prev")
   (if (null list_point)
       (setq first_point *ru_last_end_point*
               list_point (cons first_point list_point))
       (setq list_point  (cdr list_point) first_point (car list_point)))
 )
  (T;; Указана точка
   (if first_point ((eval quoted_func) first_point next_point))
   (setq list_point (cons next_point list_point) first_point next_point)
 )
)
)
(setq *ru_last_end_point* (car list_point))
)
```

Функция возвращает конечную точку последнего сегмента и сохраняет ее в глобальную переменную для последующего использования. Хитроумные функции рисования сегментов позволяют быстро наполнять меню программами рисования самых разнообразных изображений (рис. 22.4, 22.5).

Приведем еще один пример. Во многих наших программах используются заменители команды PLINE (ПЛИНИЯ). Внутри этих функций применяются низкоуровневая функция `ru-trass-draw-lw` (листинг 22.74). Этой функции передаются следующие аргументы:

- `msg` — часть приглашающего сообщения, например "трассы автодороги";
- `first_point` — первая точка трассы, или `nil`, если не задана;
- `second_point` — вторая точка трассы, или `nil`, если не задана;

- `is_closed` — флаг замыкания трассы;
- `arc_enabled` — флаг возможности рисования дуговых сегментов;
- `can_change_lw` — флаг допустимости изменения ширины линии;
- `lineweight` — вес линии;
- `pline_width` — ширина линии;
- `min_segm_length` — минимальная длина сегмента.

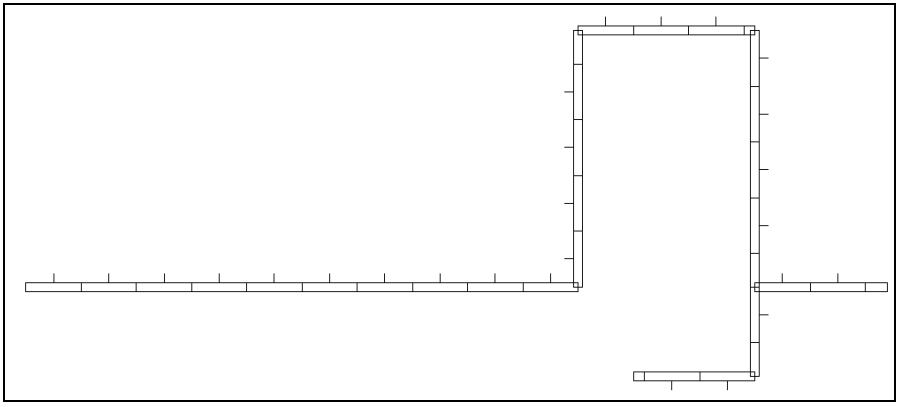


Рис. 22.4. Забор деревянный на фундаменте (обозначение на топографических планах)

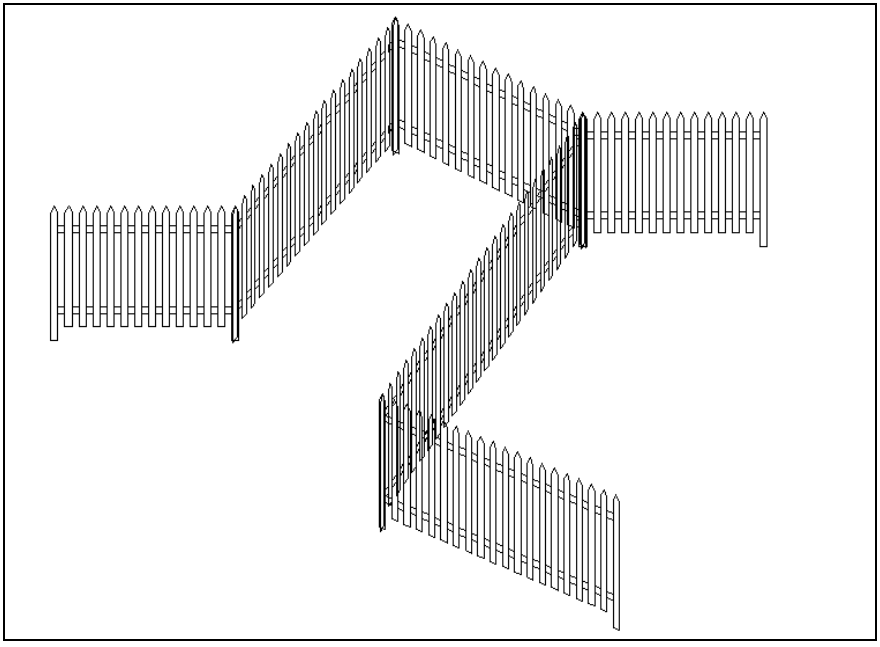


Рис. 22.5. Забор из штакетника на фасаде

Листинг 22.74. Функция _ru-trass-draw-lw

```
(defun _ru-trass-draw-lw (msg first_point second_point is_closed arc_enabled
can_change_lw lineweight pline_width min_segm_length  /
 arc_mode list_arc_mode list_point next_point nundo old_lweight old_snapang
list_snapang old_pline_width _segm_len_ok)
;;--------------------------------------------------------------------
;;; Локальная функция проверки длины сегмента
 (defun _segm_len_ok (/ result)
 (if min_segm_length
  (if (not (setq result (>= (distance first_point next_point)
              min_segm_length)));_ end of not
   (ru-msg-alert (strcat "Слишком короткий сегмент для " msg ": "
                 (rtos (distance first_point next_point))
                 "\nМинимальная длина: " (rtos min_segm_length)))
 )(setq result T)) result)
;;--------------------------------------------------------------------
  (setq old_snapang (getvar "SNAPANG") old_lweight (getvar "CELWEIGHT")
        old_pline_width (getvar "PLINEWID") nUndo 0)
  (setvar "CELWEIGHT" lineweight)
  (if (null first_point)
    (if can_change_lw
      (while
        (not (ru-is-point (setq first_point
           (ru-get-point-or-lw-or-exit (strcat "Первая точка " msg))))))
      (setq first_point
     (ru-get-point-or-exit (strcat "Первая точка " msg) nil))
   )
 )
  (if first_point
    (progn
      (setq first_point (trans first_point 0 1))
      (if (and first_point (null second_point))
        (progn
          (setq list_point    (cons first_point list_point)
                list_arc_mode (cons arc_mode list_arc_mode)
                next_point (trans (ru-get-point-reguired
                                  (strcat "Вторая  точка " msg)
                                  (trans first_point 1 0)) 0 1))
          (if (_segm_len_ok)
            (progn
              (setvar "SNAPANG" (angle first_point next_point))
              (ru-var-clear-osnap)
              (command "_.PLINE" first_point "_Width" pline_width
                pline_width next_point)
              (ru-var-restore-osnap)
              (setq nUndo (1+ nUndo)
                    list_point (cons next_point list_point)
                    list_arc_mode (cons arc_mode list_arc_mode)
                    list_snapang (cons (getvar "SNAPANG") list_snapang)
                    first_point next_point))
```

```
            (progn
              (ru-var-clear-osnap)
              (command "_.PLINE" first_point "_Width" pline_width
                  pline_width)
              (ru-var-restore-osnap))))
        (progn
;;; заданы две первые точки
          (setq next_point    second_point
                list_point    (cons first_point list_point)
                list_arc_mode (cons arc_mode list_arc_mode))
          (setvar "SNAPANG" (angle first_point next_point))
          (ru-var-clear-osnap)
          (command "_.PLINE" first_point "_Width" pline_width
                pline_width next_point)
          (ru-var-restore-osnap)
          (setq nUndo (1+ nUndo)
                list_point (cons next_point list_point)
                list_arc_mode (cons arc_mode list_arc_mode)
                list_snapang (cons (getvar "SNAPANG") list_snapang)
                first_point next_point)))
      (while (setq next_point
              (_ru-get-with-default (strcat "\nСледующая точка " msg)
               (if is_closed
                  (if (> (length list_point) 2) "Замкни" nil)
                        "Выход")
              'getpoint
              (if (and is_closed (< (length list_point) 3)) 1 nil)
              (if arc_enabled
                (if arc_mode
                  (if (> nundo 1) "Отмени Линия" "Линия")
                  (if (> nundo 1) "Отмени Дуга" "Дуга")
                )
                (if (> nundo 1) "Отмени" nil)
            )
               first_point
             );_ end of _ru-get-with-default
            );_ end of setq
        (cond
          ((= next_point "Дуга") (command "_ARC") (setq arc_mode T))
          ((= next_point "Линия") (command "_LINE") (setq arc_mode nil))
          ((= next_point "Отмени")
           (if (> nUndo 1)
             (progn
               (command "_UNDO")
               (setq nUndo         (1- nUndo)
                    ;; удалить из списка точек последнюю
                     list_point    (cdr list_point)
                     first_point   (car list_point)
                     list_arc_mode (cdr list_arc_mode)
                     arc_mode      (car list_arc_mode)
                     list_snapang  (cdr list_snapang)
              );_ end of setq
```

```
                (setvar "SNAPANG" (car list_snapang))
              );_ end of progn
              (progn
                (ru-msg-alert "Больше нечего отменять!")))
          )
           (T;указана точка
            (if (_segm_len_ok)
              (progn
                (setvar "SNAPANG" (angle first_point next_point))
                (setq first_point next_point
                      list_point (cons next_point list_point)
                      list_arc_mode (cons arc_mode list_arc_mode)
                      list_snapang (cons (getvar "SNAPANG") list_snapang)
                      nUndo (1+ nUndo)) (command next_point))))
       );_ end of cond
     );_ end of while
      (if is_closed (command "_Close") (command ""))
   );_ end of progn
 );_ end of if
  (setvar "SNAPANG" old_snapang)
  (setvar "CELWEIGHT" old_lweight)
  (setvar "PLINEWID" old_pline_width)
  list_point
)
```

Как видно из исходного текста, функция имитирует команду PLINE (ПЛИНИЯ) с некоторыми упрощениями. Все сложности вызваны необходимостью ведения списков истории различных режимов для возможности отмены нарисованных сегментов. Рисование производится с помощью незавершаемой до последнего момента функции `command`, поэтому внутри функции используются координаты точек в ПСК и производятся преобразования при передаче и получении точек в функции, работающие в МСК.

Приведем примеры использования функции `_ru-trass-draw-lw` в среднеуровневых функциях. Их кажущееся чрезмерное разнообразие (а может быть, наоборот, единообразие) вызвано широким кругом решаемых задач, каждая из которых немного отличается от другой. Когда-то все эти задачи были реализованы в автономных программах, но мы, руководствуясь проводимой в жизнь идеологией, попытались объединить их одной базовой функцией. Итак, представляем это "отвратительное сборище" так называемых "функций". Первая из них (листинг 22.75) всего лишь вызывает базовую функцию, но с заданной первой точкой, и возвращает последнюю точку нарисованной линии. Так где-то было надо.

Листинг 22.75. Функция `ru-pline-draw-closed-lw-msg-from-pt1`

```
(defun ru-pline-draw-closed-lw-msg-from-pt1 (msg first_point can_change_lw
lineweight pline_width / list_point)
 (if
(setq list_point (_ru-trass-draw-lw msg first_point nil t NIL
    can_change_lw  lineweight pline_width  NIL))
```

```
    (car list_point) nil
 );_ end of if
)
```

Вторая модификация (листинг 22.76) рисует замкнутый контур, запрашивая все точки. Возвращает список вершин нарисованных сегментов. Рисование дуг не допускается.

Листинг 22.76. Функция ru-pline-draw-closed-lw-msg

```
(defun ru-pline-draw-closed-lw-msg (msg can_change_lw lineweight pline_width)
  (_ru-trass-draw-lw
    msg nil nil t nil can_change_lw lineweight pline_width nil)
)
```

Третий вариант (листинг 22.77) рисует замкнутый контур "полосы отвода" с заданной шириной "защитной зоны". Знающие люди понимают, как часто это нужно делать. Возвращается список координат созданного замкнутого контура.

Листинг 22.77. Функция ru-pline-make-trace-list-vert-closed-by-axis-draw

```
(defun ru-pline-make-trace-list-vert-closed-by-axis-draw
       (trace_name trace_half_width lineweight / list_point)
;;; создание полосы трассы  рисованием осевой линии
;;; делает замкнутую полилинию
;;; Параметры
;;; trass_name - название трассы
;;; trace_half_width - полуширина трассы
;;; возвращает список координат новой полилинии
 ;| Пример:
(ru-pline-make-trace-list-vert-closed-by-axis-draw "полоса отчуждения" 50.0 0)
 |;
  (if (setq list_point
    (_ru-trass-draw-lw (strcat "осевой линии объекта '" trace_name "'")
     nil nil nil nil nil lineweight 0 nil))
    (setq list_point
      (ru-pline-make-trace-draw-closed-by-axis (entlast) (car list_point)
             (angle (cadr list_point) (car list_point)) trace_half_width
             T lineweight)
   );_ end of setq
 );_ end of if
  list_point
)
```

Четвертая производная функция отличается от базовой тем, что возвращает угол последнего сегмента (листинг 22.78).

Листинг 22.78. Функция ru-trass-draw-lw-from-pt1-arc

```
(defun ru-trass-draw-lw-from-pt1-arc (msg can_change_lw lineweight
pline_width / list_point)
```

```
 (if (setq list_point
            (_ru-trass-draw-lw
              msg nil nil nil arc_enabled can_change_lw lineweight
               pline_width nil)
     );_ end of setq
    (angle (cadr list_point) (car list_point)) nil))
```

Следующий вариант (листинг 22.79) отличается от предыдущего тем, что задана первая точка трассы, и рисование начинается с запроса следующей точки.

Листинг 22.79. Функция `ru-trass-draw-lw-from-pt2`

```
(defun ru-trass-draw-lw-from-pt2 (msg first_point lineweight
    pline_width / list_point)
(if (setq list_point
    (_ru-trass-draw-lw
       msg first_point nil nil nil nil lineweight pline_width nil))
    (angle (cadr list_point) (car list_point)) nil))
```

Очередной вариант (листинг 22.80) отличается тем, что запрос начинается с третьей точки, да еще имеется ограничение на минимальную длину сегмента. Сантехники сразу сообразят, что дает им такой вариант — рисование гнутых трубопроводов с ограничением минимального радиуса гнутья.

Листинг 22.80. Функция `ru-trass-draw-lw-from-pt3-min-segm`

```
(defun ru-trass-draw-lw-from-pt3-min-segm (msg first_point second_point
 lineweight pline_width min_segm_length / list_point)
  (setq list_point
     (_ru-trass-draw-lw
       msg first_point second_point nil nil nil lineweight pline_width
          min_segm_length)) (angle (cadr list_point) (car list_point)))
```

Вариант, представленный в листинге 22.81, позволяет рисовать от заданной первой точки трассу *текстовым типом линии*. Текстовый тип линии автоматически создается из заданного набора символов — короткой строки, соответствующей буквенно-цифровым обозначениям трубопроводов и трасс или горизонталей вертикальной планировки (Т1, В1.2, 100.50 и т. п.). Как автоматически создавать текстовые типы линий, мы рассмотрим чуть ниже. Использование текстового типа линии имеет большое достоинство — вся линия является единым примитивом, но имеет и большой недостаток — текст может оказаться написанным и "вверх ногами". Текстовый тип линии можно формировать не только из букв и цифр, но и из других клавиатурных символов, что позволяет получить интересные эффекты.

Листинг 22.81. Функция `ru-trass-draw-lw-txt`

```
(defun ru-trass-draw-lw-txt (txt len_dash first_point second_point
arc_enabled can_change_lw lineweight pline_width / ltype old_ltype)
```

```
;|Рисование линии текстовым типом линии. Аргументы:
txt - символ
arc_enabled - разрешены дуги или NIL
len_dash - длина пунктира или NIL при сплошной линии
first_point, second_point - точки начала и конца или NIL для многосегментной
Пример:
 (ru-trass-draw-lw-txt "B1" 10 nil nil T T (ru-lw-current) 0)
|;
 (setq old_ltype (getvar "CELTYPE"))
 (if (setq ltype (ru-ltype-make-txt txt len_dash))
  (progn
   (ru-ltype-set-current ltype)
    (setq list_point
      (_ru-trass-draw-lw "линии" first_point second_point nil
         arc_enabled can_change_lw lineweight pline_width nil
      ))
     (ru-ltype-set-current old_ltype)
   )
    (princ "\nНе могу создать такой тип линии!")
 )(princ))
```

Функция `ru-pline-measure-block` (листинг 22.82) разбивает нарисованную трассу заданным блоком, создаваемым из заданного единичного блока.

Листинг 22.82. Функция `ru-pline-measure-block`

```
(defun ru-pline-measure-block (prmpt measure_dist unit_block
      new_block_size_mm new_block_rotate first_point / block_name
      list_point)
;;;  prmpt - приглашение для  запроса  шага NIL, если задан
;;;  unit_block   - блок для разметки
;;;  new_block_size_mm    - размер разметочного  блока, ММ НА БУМАГЕ
;;;  new_block_rotate     - угол наклона  разметочного блока
;;;  measure_dist - шаг  разметки по  умолчанию, ед. чертежа,
;;;  first_point - первая точка,Nil - если не задана
 (setq block_name (ru-block-make-for-measure unit_block new_block_size_mm
           new_block_rotate)
 (if prmpt ;если задано приглашение
 (setq measure_dist
  (ru-conv-meter-to-unit (atof (ru-user-read-last-param prmpt "3.0")))
      measure_dist (ru-get-dist prmpt measure_dist nil))
    (setq measure_dist (abs measure_dist))
 )
  (if (setq list_point
   (_ru-trass-draw-lw " на оси трассы  " first_point nil nil T T
     (ru-lw-current) 0 nil))
    (progn
      (if (> (ru-geom-pline-length list_point nil T) measure_dist)
        (progn
          (ru-var-clear-osnap)
```

```
          (command "_.MEASURE" (list (entlast) (car list_point))
              "_B" block_name "" measure_dist)
          (ru-var-restore-osnap)
       )
     );_ end of if
      (if prmpt
        (ru-user-write-last-param prmpt
         (rtos (ru-conv-unit-to-meter measure_dist) 2 2)))
   )) (princ))
```

Создание замкнутой "полосы отвода" по примитиву "оси полосы" производится функцией, приведенной в листинге 22.83.

Листинг 22.83. Функция `ru-pline-make-trace-draw-closed-by-axis`

```
(defun ru-pline-make-trace-draw-closed-by-axis (ent_axis last_axis_pnt
last_axis_angle half_width is_delete_axis lineweight / lst lst_vertex lst_vertex1)
;| Параметры
ent_axis         - осевая линия (примитив)
last_axis_pnt    - последняя точка осевой
last_axis_angle   - угол последнего сегмента осевой
half_width       - полуширина трассы от осевой
is_delete_axis   - T - стирать ли осевую линию
Возвращает список координат новой полилинии
|;
(if
 (vl-cmdf "_.OFFSET" half_width (list ent_axis last_axis_pnt)
  (polar last_axis_pnt (ru-geom-go-right last_axis_angle) half_width) "")
  (if (setq lst (ru-pline-list-vertex (entlast)))
   (progn
    (setq lst_vertex1 (car lst)) (entdel (entlast))
    (if
      (vl-cmdf "_.OFFSET" half_width (list ent_axis last_axis_pnt)
       (polar last_axis_pnt (ru-geom-go-left last_axis_angle)
                        half_width) "")
      (if (setq Lst (ru-pline-list-vertex (entlast)))
        (progn
          (setq lst_vertex (append lst_vertex1 (reverse (car Lst))))
          (entdel (entlast))
          (ru-pline-add lst_vertex t 0 lineweight nil)
          (if is_delete_axis (entdel ent_axis))
   ))))))) lst_vertex)
```

Следующая функция (листинг 22.84) создает определение блока, используемого в функции `ru-pline-measure-block`. Имя создаваемого блока включает основные параметры вставки, т. к. для разных условий могут понадобиться разные производные блоки из одного базового. В этой функции мы создадим новый блок с помощью функции `entmake`.

Листинг 22.84. Функция `ru-block-make-for-measure`

```
(defun ru-block-make-for-measure (unit_block new_block_size_mm new_block_rotate /
new_block_size_unit new_block_name)
;|
Аргументы:
unit_block - единичный блок, определенный в рисунке
new_block_size_mm - размер генерируемого блока в мм на бумаге
new_block_rotate — угол наклона блока по отношению к оси трассы
|;
 (if (tblsearch "block" unit_block)
  (progn
   (setq new_block_size_unit
        (ru-conv-millimeter-in-paper-to-unit new_block_size_mm)
         new_block_name
            (strcat unit_block (rtos (ru-scale-current-space) 2 0) "_"
             (rtos new_block_rotate 2 0) (ru-unit-name))
  )
   (if (not (tblsearch "block" new_block_name))
        (mapcar 'entmake
           (list (list '(0 . "BLOCK") '(100 . "AcDbEntity")
              '(100 . "AcDbBlockBegin") (cons 2 new_block_name)
              '(10 0.0 0.0 0.0) '(62 . 256) '(8 . "0") '(70 . 0))
           (list '(0 . "INSERT") '(100 . "AcDbEntity")
             '(100 . "AcDbBlockReference") (cons 2 unit_block) '(8 . "0")
             '(10 0.0 0.0 0.0) '(62 . 0) (cons 41 new_block_size_unit)
              (cons 42 new_block_size_unit) (cons 43 new_block_size_unit)
              (cons 50 new_block_rotate))
           (list '(0 . "ENDBLK") '(100 . "AcDbEntity")
             '(100 . "AcDbBlockEnd") '(8 . "0"))))
  new_block_name
 )
 nil
)
)
```

Программная работа с типами линий

Теперь разработаем функцию для программного создания текстового типа линии, используемого в функции `ru-trass-draw-lw-txt`. В аргументах функции (листинг 22.85) мы будем передавать текст и длину пунктиров в миллиметрах на бумаге, если требуется "текстово-пунктирная линия". При сплошной линии длина пунктира `nil`. Функция кажется простой, т. к. разбита на несколько простых функций.

Листинг 22.85. Функция `ru-ltype-make-txt`

```
(defun ru-ltype-make-txt (txt len_dash / lin_file_name ltype_def_list
 ltype_name _make-txt-dashed-ltype-def-list _make-txt-ltype-def-list
 _text-def-string _text-header-string _line-header-string _add-ltype
```

```
 _make-dashed-ltype-def-list _make-txt-dashed-ltype-name
 _make-txt-ltype-name _make-dashed-ltype-name _normal-ltype-name)
;;; Создание текстового типа линии
;;; txt - символ
;;; len_dash - длина пунктира или NIL при сплошной

;;;------- Локальная функция--------------------------
  (defun _normal-ltype-name (string)
   ;; независимо от EXTNAMES заменяем недопустимые символы
    (strcat
      (vl-string-translate " <>/\\\":?*|,=`;." "_lrbsq820Iz-adt" string)
      "_RU"))

;;;------- Локальная функция--------------------------
  (defun _make-dashed-ltype-name (len_dash)
   (strcat "DASH_" (itoa len_dash) "x1"))

;;;------- Локальная функция--------------------------
  (defun _make-txt-ltype-name (txt) (strcat "TXT_" txt))

;;;------- Локальная функция--------------------------
  (defun _make-txt-dashed-ltype-name (txt len_dash)
    (strcat (_make-txt-ltype-name txt) "_"
            (_make-dashed-ltype-name len_dash)))

;;;------- Локальная функция--------------------------
  (defun _add-ltype (lin_file_name ltype_name ltype_def_list / file)
    (if (setq file (open lin_file_name "a"))
      (progn
        (princ "\n;; ruCAD auto-created linetype\n" file)
        (princ (car ltype_def_list) file)
        (princ (cadr ltype_def_list) file)
        (setq file (close file))
       ltype_name) nil))

;;;------- Локальная функция--------------------------
  (defun _make-dashed-ltype-def-list (ltype_name len_dash)
   (list
    (strcat "*" ltype_name "," (repeat len_dash "_")
                           " " (repeat len_dash "_") "\n")
    (strcat "A," (itoa len_dash) ",-1\n")))

;;;------- Локальная функция--------------------------
  (defun _line-header-string (dashed)
    (if dashed "- - -" "_____"))

;;;------- Локальная функция--------------------------
  (defun _text-header-string (ltype_name txt dashed)
    (strcat "*" ltype_name "," (_line-header-string dashed) txt
       (_line-header-string dashed) "\n"))
```

```
;;;------- Локальная функция--------------------------
  (defun _text-def-string (txt)
    (strcat ",-1,[" (chr 34) txt (chr 34) ",RU_LINE,S=2,R=0,X=0,Y=-1],-"
      (rtos (* (strlen txt) 2) 2 2)))

;;;------- Локальная функция--------------------------
  (defun _make-txt-ltype-def-list (ltype_name txt)
   ;; Текст без пунктира
    (list (_text-header-string ltype_name txt nil)
          (strcat "A,10" (_text-def-string txt) ",10\n")))

;;;------- Локальная функция--------------------------
  (defun _make-txt-dashed-ltype-def-list (ltype_name txt len_dash)
   ;; если не найден, сочиняем строку
    (list (_text-header-string ltype_name txt t)
      (strcat "A," (itoa len_dash) ",-1," (itoa len_dash)
        (_text-def-string txt) "\n")))

;;;------- Главная функция---------------------------
  (princ "\nСоздаю тип линии...")
  (setq lin_file_name (ru-file-acad "rucad.lin" t))
;;; формируем имя линии
  (cond
;;; Недопустимое сочетание
    ((and (not len_dash) (= txt "")) (setq ltype_name NIL))
;;; Пунктир без текста
    ((and len_dash (= txt ""))
     (setq len_dash (ru-match-round len_dash)
       ltype_name
          (_normal-ltype-name (_make-dashed-ltype-name len_dash))
       ltype_def_list
            (_make-dashed-ltype-def-list ltype_name len_dash)))
   ;; Текст без пунктира
    ((and (not len_dash) (/= txt ""))
     (setq ltype_name (_normal-ltype-name (_make-txt-ltype-name txt))
           ltype_def_list (_make-txt-ltype-def-list ltype_name txt)))
   ;; И пунктир и текст
    (T
     (setq len_dash (ru-match-round len_dash)
       ltype_name (_normal-ltype-name
           (_make-txt-dashed-ltype-name txt len_dash))
       ltype_def_list
           (_make-txt-dashed-ltype-def-list ltype_name txt len_dash)))
 );_ end of cond
;;; проверяем наличие такого описания
  (if ltype_name
    (if (not (ru-ltype-find-in-lin lin_file_name ltype_name))
      (setq ltype_name
          (_add-ltype lin_file_name ltype_name ltype_def_list))
   );_ end of if
 );_ end of if
  ltype_name
)
```

Поиск описания линии в LIN-файле выполняет функция, показанная в листинге 22.86.

Листинг 22.86. Функция `ru-ltype-find-in-lin`

```
(defun ru-ltype-find-in-lin (lin_file_name line_name / s1 file result)
(if (setq file (open lin_file_name "r"))
 (progn
  (while (and (setq s1 (read-line file)) (null result))
   (setq result
    (= (substr s1 1 (+ (strlen line_name) 2))
       (strcat "*" line_name ",")))
 );_ end of while
  (setq file (close file))
);_ end of progn
);_ end of if
 result
)
```

Загрузку заданного типа линии с перебором файлов ruCAD.lin и acadiso.lin выполняет функция, приведенная в листинге 22.87.

Листинг 22.87. Функция `ru-ltype-load`

```
(defun ru-ltype-load (linetype)
  (defun _ltype-load (linetypename filename)
    (if (and (not (ru-ltype-exists linetypename))
     (vl-catch-all-error-p (vl-catch-all-apply 'vla-load
       (list (ru-ltype-get-ltypes) linetypename filename))))
      nil T))
  (cond
    ((_ltype-load linetype (ru-file-acad "ruCAD.lin" T)) linetype)
    ((_ltype-load linetype (findfile "acadiso.lin")) linetype)
    (T nil))
)
```

Установку активного типа линии выполняет функция, приведенная в листинге 22.88.

Листинг 22.88. Функция `ru-ltype-set-current`

```
(defun ru-ltype-set-current (Ltype)
 (cond
  ((ru-ltype-load Ltype)
     (vlax-put-property (ru-obj-get-active-document) "ActiveLineType"
       (vla-item (ru-obj-doc-collection "Linetypes") ltype))
 )
  (T (princ (strcat "\nНевозможно установить тип линии " Ltype
              "!\nУстанавливаю СПЛОШНУЮ линию!"))
```

```
     (vlax-put-property (ru-obj-get-active-document) "ActiveLineType"
       (vla-item (ru-obj-doc-collection "Linetypes") "CONTINUOUS")))
 )
 (getvar "CELTYPE")
)
```

Загрузка программ

Теперь нам предстоит разработать самую главную функцию, без которой ничего работать не будет — функцию загрузки наших собственных приложений. Использовать обычный вариант (load "имя_файла") можно только в простейших случаях. Прежде чем вызвать функцию load, мы должны о многом позаботиться, в том числе о собственной работе (листинг 22.89).

Напомним нашу стратегию — библиотека функций (с именами ru-*) всегда загружена при запуске системы, а функции-программы (с именами ru_*) загружаются *каждый раз* при выполнении выражения (ru-app-load "короткое_имя_программы").

Листинг 22.89. Функция ru-app-load

```
(defun ru-app-load (name / lsp_src version result _ru-app-load-lsp)
;|-------------------------------------------------------------------------
Локальная функция загрузки файла из каталога App
-------------------------------------------------------------------------|;
(defun _ru-app-load-lsp (name ext / lsp result)
 (setq lsp (strcat (ru-file-app name) ext))
 (if (findfile lsp)
  (if (equal (load lsp "Failed") "Failed")
   (princ (strcat "\nПрограмма \n" lsp
      "\nне может быть загружена! Возможно, она испорчена!"))
    (setq result t)))
    result
 )
;|-------------------------------------------------------------------------
Основная функция. В режиме предварительного просмотра справки перед загрузкой
программы выводится для просмотра текстовый файл со справкой.
Если одновременно включен режим *ru_developer* и пользователь имеет право
редактировать текстовые справки, файл открывается в режиме редактирования. Это
позволяет одновременно с тестированием программ "по горячим следам" корректировать
справки.
-------------------------------------------------------------------------|;
 (if *ru_previewhelp*
  (ru-help-txt-view (ru-file-help (strcat "txt\\" name ".txt"))
   "О программе" (if *ru_developer* name "")
     (and *ru_developer* (ru-user-may-txt-hlp-edit)))
;|-------------------------------------------------------------------------
Закомментированы возможные варианты просмотра HTML-справки и "Советов дня"
-------------------------------------------------------------------------|;
;;; Вариант с HTML-файлом
;;;(ru-help-view name (and *ru_developer* (ru-user-may-develop)))
```

```
;;; Вариант с "Советами дня"
;;; (ru-dlg-tips (ru-user-long-name) "А знаете ли вы, что..."
;;; (strcat (ru-dirs-get-local-app-data) "TipsOfDay\\" name ".tips"))
)
;|-------------------------------------------------------------------------
В режиме *ru_developer* при запущенной IDE Visual Lisp разыскивается исходный текст
программы, перекомпилируется, FAS-файл отправляется в каталог App. Это позволяет
разработчику избежать нудных операций копирования в
черт_знает_где_расположенный_каталог.
-------------------------------------------------------------------------|;
 (if (and *ru_developer* (dos_isvlide))
  (progn
    (setq lsp_src (ru-file-set-ext (ru-file-lsp name) ".lsp"))
    (if (findfile lsp_src) (ru-app-make-fas-file name)
      (princ (strcat "\nНе найден исходный текст " lsp_src "\n"))))
  (setq result T)
;|-------------------------------------------------------------------------
Сначала пытаемся загрузить FAS-файл.
-------------------------------------------------------------------------|;
  (if (not (_ru-app-load-lsp name ".fas"))
;|-------------------------------------------------------------------------
Если FAS-файл не загружен, ищем рядом с ним LSP-файл. Это чрезвычайная ситуация,
LSP-файлов там быть не должно, но иногда разработчикам приходится разбираться
на компьютерах пользователей с непонятными проблемами, и в это время они могут
принести исходники с собой. Возможен также вариант с размещением в каталоге App
именно исходных файлов.
-------------------------------------------------------------------------|;
  (if (not (_ru-app-load-lsp name ".lsp"))
;|-------------------------------------------------------------------------
Если вообще не удалась загрузка, и файлы не найдены, то выводим благопристойное
сообщение, благодаря которому пользователь думает, что это он виноват, что приобрел
какой-то ограниченный комплект. На самом же деле это просто мы эту программу
где-то потеряли или еще не сделали.
-------------------------------------------------------------------------|;
   (progn (ru-msg-alert (strcat "Программа " name
          " не входит в состав комплекта '" (ru-rucad-version) "'"))
       (setq result nil))))
  result
)
```

Загрузочная функция довольно часто модифицируется, в нее добавляются и удаляются загрузки ARX- и VLX-приложений, порядок вывода и формат справок.

Резюме

Продолжать описание библиотечных функций мы могли бы практически бесконечно, но вынуждены остановиться на этих стратегических примерах. Пора переходить к разработке прикладных программ.

ГЛАВА 23

Итоги разработки библиотек

Как ни странно, но подводить итоги разработки библиотек еще рано. Мы можем только зафиксировать итоги глав книги, посвященных разработке библиотек. Сами библиотеки функций будут непрерывно изменяться во время всего жизненного цикла программ. Даже в последующих главах книги мы неоднократно будем упоминать о том, что нам приходится изменять (иногда радикально) ранее разработанные функции.

Не следует рассматривать приведенные нами решения как единственно верные и вечные — это всего лишь возможные, иногда не самые лучшие варианты.

Мы сумели показать только небольшую часть исходных текстов используемых нами функций — в основном таких, которые демонстрируют последовательную поддержку концептуальных решений, выработанных при постановке задачи.

Мы также показали, как интересная идея — программное управление доступом к меню — при правильной (надеемся) реализации в наших функциях должна быть отвергнута из-за какой-то внутренней ошибки реализации методов в самой системе AutoCAD. В результате нам пришлось защищать пункты меню с использованием выражений языка DIESEL и системной переменной.

Используемый нами подход — много мелких функций, из которых должны собираться, по возможности, маленькие по объему, конечные программы, удобен при разработке большой системы. Постороннему человеку трудно разобраться в большом количестве функций, но это и не нужно. Наши библиотеки предназначены не только для внутреннего употребления, ими могут воспользоваться и пользователи нашей системы для разработки собственных программ. Мы постарались сделать так, чтобы большую часть адаптации нашей системы можно было выполнять только на уровне меню. Для этого надо знать уже не несколько сотен наших функций, а всего лишь несколько штук. Если же прикладной программист захочет разработать более серьезные программы, то ему, конечно, потребуется изучить библиотеки детально, т. е. использовать как бы новый диалект LISP. При этом может оказаться, что, применяя нашу функцию, программист на самом деле через цепочку посредников обращается к низкоуровневым функциям Windows API.

При разработке библиотек мы также максимально использовали опыт других разработчиков. Например, мы очень часто использовали библиотеку JEDI, хотя разобраться с огромным количеством функций этой библиотеки иногда было сложнее,

чем написать свои аналоги. Однако мы предполагали, что авторы JEDI лучше нас знакомы со многими деталями, и такие предположения обычно оправдывались.

Итак, в итоге разработки библиотек мы имеем огромный набор функций для решения практически любых требуемых нам задач. А если чего-то еще нам не хватит, то мы уже владеем методикой и некоторыми новыми технологиями.

В последующих главах мы займемся разработкой прикладных программ, т. е. функций, к которым имеет доступ конечный пользователь посредством интерфейсов нашей системы.

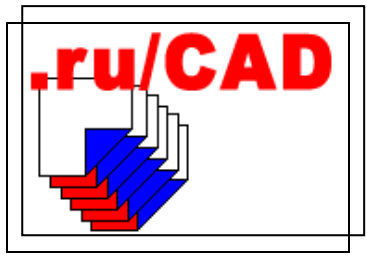
От первоначального плана демонстрации конкретных решений для разных разделов проекта нам пришлось отказаться из-за незапланированного роста объемов предыдущих глав. Например, *глава 10*, в которой мы намечали на десяти страницах дать краткий обзор самых грубых ошибок, "разбухла" в десять раз за счет подробного обзора технологий. Более подробно, чем предполагалось, мы рассмотрели практически все малоизученные темы — COM-технологии, XML-меню, работу с базами данных.

В дальнейшем мы так же будем сосредотачиваться на менее известных решениях, а от разбора "рисовальных" программ практически откажемся. В конце концов, разработать программу рисования любого объекта достаточно просто — нужно вычислить координаты точек и по этим координатам нарисовать изображение объекта любыми методами. Сложнее сделать надежную программу, т. е. защищенную и от неверных действий пользователя, и от недостаточного учета особенностей работы (например, побочного влияния объектных привязок), но этим вопросам мы уже уделили достаточно много внимания и места.

Некоторые прикладные программы мы уже рассмотрели, например: рисование форматов в *главе 21*, рисование таблиц в *главе 17*, работу с классификатором слоев в *главе 18*. Вызов некоторых библиотечных функций может производиться непосредственно из меню, для некоторых потребуется небольшая "обертка" для упрощения синтаксиса макросов меню.

В *части IV* книги мы рассмотрим несколько типовых или интересных прикладных программ — в основном таких, какие редко встречаются у других авторов. Кроме того, пару глав мы посвятим "беллетристике" — обсуждению некоторых спорных проблем.

ЧАСТЬ IV



Разработка прикладных программ

ГЛАВА 24

Разработка набора инструментов для программистов

Этой главой мы открываем *часть IV* нашей книги, посвященную разработке прикладных программ. По русской привычке мы "долго запрягали", разрабатывая бесчисленные вспомогательные функции. Теперь нам предстоит "быстро ехать", т. е. выдавать продукцию в виде программ для конечного пользователя. Однако, как мы уже писали в *главе 11*, первыми пользователями будем мы сами — разработчики системы. Прежде всего мы позаботимся об автоматизации собственного труда по разработке и сделаем для себя небольшой "сундучок" с инструментами разработчика. Многие из них мы уже реализовали в виде функций, осталось их вставить в меню системы. Все инструменты разработчика мы будем размещать в разделе меню **Профи**, закрывая их от доступа низкоквалифицированных пользователей, чтобы они по неосторожности и сами не "порезались", и окружающую обстановку не испортили. В этой главе мы приведем программы, которыми сами чаще всего пользуемся — и очень простые, и достаточно сложные.

Редактирование меню

На этапе разработки нам часто придется редактировать меню. Чтобы не блуждать по системе каталогов, напишем простую функцию для загрузки файла шаблона меню в текстовый редактор (листинг 24.1). Функция перегрузки меню после редактирования `ru-menu-reload` была описана в *главе 13*.

Листинг 24.1. Функция `ru-menu-edit`

```
(defun ru-menu-edit (/ reg_key editor)
;|-------------------------------------------------------------------------
 Используем встроенную функцию, т. к. не требуется запускать редактор
в модальном режиме
-------------------------------------------------------------------------|;
  (startapp  (ru-app-txt-editor) (ru-file-template "ruCAD.mnu"))
)
```

Переключатели режимов в меню

У нас имеется несколько глобальных переменных, определяющих режим работы системы. Для переключения значений этих переменных (рис. 24.1) разработаем соответствующую диалоговую функцию (листинг 24.2).

Листинг 24.2. Функция `ru-var-check-dlg`

```
(defun ru-var-check-dlg (/ var_values)
  (if (setq var_values (ru-dlg-show-check-list "Установка переключателей"
              (list "Вывод отладочных сообщений"
                    "Предварительная справка"
                    "Редактирование XML"
                    "Режим разработчика"
              );_ end of list
              (list  *ru_msg_debug*
                     *ru_previewhelp*
                     *ru_xml_edit*
                     *ru_developer*
              );_ end of list
              (list T T (ru-user-may-xml-edit) (ru-user-may-develop))
            );_ end of ru-dlg-show-check-list
    );_ end of setq
    (mapcar 'ru-var-set-var
            (list "*ru_msg_debug*" "*ru_previewhelp*" "*ru_xml_edit*"
            "*ru_developer*")
             var_values
   );_ end of mapcar
 );_ end of if
(princ)
);_ end of defun
```

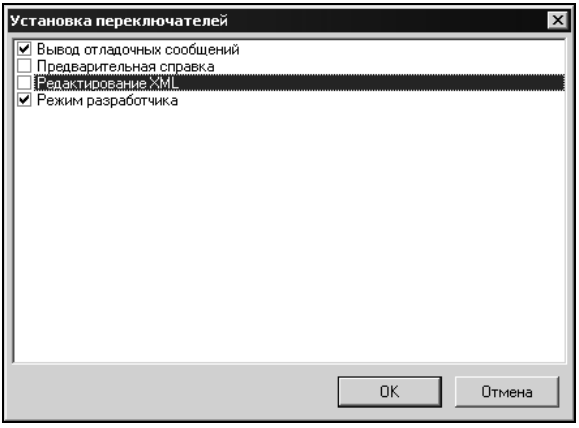


Рис. 24.1. Переключение режимов работы

Установка режима работы с запоминанием для текущего пользователя реализована в функции, показанной в листинге 24.3.

Листинг 24.3. Функция `ru-var-set-var`

```
(defun ru-var-set-var (svar value)
;;Пример (ru-var-set-var "*ru_msg_debug*" T)
 (set (read svar) value);_ end of set
 (ru-user-write-last-param svar
     (itoa (ru-conv-value-to-wordbool value)))
 value
)
```

Массовые операции с файлами

Очень часто приходится выполнять рутинные операции с большим числом примитивов. Иногда это нужно проделать и со множеством файлов. Типичным примером является конвертирование файлов, созданных в DOS-версиях системы AutoCAD. Еще одним примером может служить операция, которую нам приходилось проделывать с несколькими тысячами файлов топографических планшетов. С каждым файлом, созданным в системе AutoCAD R10, нужно было проделать такие манипуляции:

- конвертировать каждый файл в WIN-кодировку с изменением всех надписей и имен именованных объектов;
- изменить имена слоев и блоков по новому стандарту классификатора;
- "нормализовать" определения блоков и примитивов — примитивы блоков перенести на слой 0 с цветом BYBLOCK (ПОБЛОКУ), цвет всех примитивов установить BYLAYER (ПОСЛОЮ), изменить цвета слоев на стандартные;
- вычислить новое имя файла планшета в соответствии с системой разграфки планшетов;
- вычислить имена восьми примыкающих планшетов, определить, есть ли они в наличии, а если есть, то нарисовать на специальном слое рамки планшетов и привязать к рамкам гиперссылки на эти планшеты;
- изготовить для каждого планшета по два DWF-файла (с полным набором слоев для служебного пользования и с сокращенным набором слоев для публикации в Интернете);
- записать все файлы в соответствующие каталоги;
- сгенерировать HTML-файлы для публикации DWF в Интернете.

Разумеется, вручную проделать это было немыслимо, поэтому пришлось написать несколько специальных функций и единственную программу. При разработке системы ruCAD нам приходится выполнять много подобных действий. В основном это вызвано переходом на "стандарт ruCAD" по именованию слоев и блоков. Блоков и библиотек блоков мы используем много, не исключено пополнение их из посторонних источников, которые также нужно нормализировать. Изредка нам придется использовать и файлы в DOS-кодировке, потребуются и функции для конвертирования.

Методика обработки списка файлов

Для операций с несколькими файлами часто используют механизм *сценариев (пакетов)*. В ранних версиях AutoCAD сценарии были единственным способом автоматизации работы. В более поздних, например, в незабвенной системе AutoCAD R10, также были единственным средством автоматизации при выборе из "черного меню[1]", в современных версиях механизм сценариев программистами используется редко. Мы даже не упоминали его в *главе 9* при обсуждении инструментов разработчика. В некоторых ситуациях сценарии могут оказаться незаменимым инструментом (например, в AutoCAD LT). Имеется значительное количество пользователей системы AutoCAD, принципиально автоматизирующих свою работу только с помощью "штатных" команд и иных средств *адаптации* AutoCAD, включая сценарии[2]. Недостаток механизма сценариев проистекает из его преимущества — сценарий представляет собой записанную в файл последовательность инструкций, которые должны быть введены в командной строке для выполнения задачи. Заметим именно абсолютно точную и правильную последовательность, не допускающую ни одного лишнего или недостающего символа, да еще с учетом того, что при любой ошибке или неправильно предсказанной ситуации работа сценария будет прервана. Добиться этого несложно в простых ситуациях[3], а для "хитрых" задач требуется и "хитрый" сценарий. Разумеется, сценарий можно сгенерировать, но в этом случае проще разработать программу, которая просто будет выполнять требуемые действия, но с использованием всей мощи систем программирования.

Про сценарии мы упомянули потому, что разработчики часто не догадываются, как из программы, запущенной в адресном пространстве одного документа, обработать другие документы — именно в такой ситуации вспоминают про язык сценариев. Однако задача решается достаточно просто и можно написать единственную базовую функцию (листинг 24.4), с помощью которой выполняются любые операции с любым количеством файлов.

Функции передаются аргументы:

- `first_message` — предупреждающее сообщение перед запуском массовых операций или `nil`, если предупреждения не требуется;
- `files_list` — список полных имен обрабатываемых файлов;
- `test_func` — имя тестовой функции, выводящей сообщение `first_message` и проверяющей возможность начала обработки файлов по списку;
- `quoted_func` — имя функции, выполняющей операцию с именем одного файла из списка.

Листинг 24.4. Функция `ru-batch-file-operations`

```
(defun ru-batch-file-operations (first_message files_list test_func
                                               quoted_func / do_it)
```

[1] Главное меню AutoCAD ранних версий, выводившееся в текстовом режиме. Пункты этого меню выбирались вводом номера пункта.

[2] Сторонникам такого подхода рекомендуем очень интересную книгу: Свет В. Л. AutoCAD: язык макрокоманд и создание кнопок. — СПб.: БХВ-Петербург, 2004.

[3] Очень популярны были сценарии с единственной строкой, в которой был записан символ 1 или 2 — для действий из "черного меню".

```
  (if (not first_message)
    (setq do_it t)
    (setq do_it ((eval test_func) first_message))
 );_ end of if
  (if do_it (mapcar quoted_func files_list))
  (princ)
);_ end of defun
```

С файлами могут производиться любые операции, не обязательно требующие открытия самих файлов. Вот пример использования, не имеющий практического смысла, но иллюстрирующий механизм работы функции:

```
(ru-batch-file-operations (strcat "Вывод списка файлов"
"\nПОЛЬЗОВАТЬСЯ ОСТОРОЖНО!\n\nБудем делать")
(list "c:\\.ru\\ru-lib-bolt.dwg""c:\\.ru\\ru-lib-build.dwg"
 "c:\\.ru\\ru-lib-car.dwg")
'ru-yes 'print)
```

В этом примере основной функцией, выполняющей операции, является `print`, т. е. в этом случае будут просто выведены имена файлов. Тестовая функция `ru-yes` выведет предупреждающее сообщение, и если будет нажата кнопка **Да**, то опасная операция печати списка будет выполнена.

Разумеется, нам нужны функции, делающие что-то полезное, и их можно разработать сколько угодно, главное, чтобы в момент выполнения функции `ru-batch-file-operations` и тестовая, и основная функции были определены. Далее разберем пример программы для массовых операций над файлами — "нормализацию" библиотек блоков.

Нормализация файлов

Мы договаривались, что блоки, используемые в системе ruCAD, должны состоять из примитивов, находящихся на слое 0 и имеющих цвет ByBlock (ПОБЛОКУ), а все тексты и атрибуты должны быть созданы стилем RU_CAD. Блоков у нас очень много, и в отдельных файлах, и в библиотеках, поэтому требуется программа для приведения их в "нормальное" состояние (листинг 24.5).

Листинг 24.5. Файл ru_pro_batch_normal.lsp

```
(defun START (/ files_list)
  (if (ru-user-action-enabled "DEVELOP" T)
    (progn
      (if (setq files_list (ru-dlg-file-select-dwg-multi
                  "Выбор файлов для нормализации" ""
                  (ru-user-read-last-param "pro_batch_normal_dir"
                    (ru-user-work-dir)
                  );_ end of ru-user-read-last-param
                  (ru-user-action-enabled "DEVELOP" nil)
                );_ end of ru-dlg-file-select-dwg-multi
        );_ end of setq
```

```
        (progn
          (ru-user-write-last-param "pro_batch_normal_dir"
            (vl-filename-directory (car files_list))
         );_ end of ru-user-write-last-param
          (ru-batch-file-operations
            (strcat
              "Программа перерабатывает выбранные файлы."
              "\nВсе примитивы, включая вложенные в блоки,"
               " переносит на слой 0"
              "\nВсе стили переводит в RU_CAD"
              "\nПОЛЬЗОВАТЬСЯ ОСТОРОЖНО!\nТОЛЬКО ДЛЯ РИСУНКОВ, "
               "ВСТАВЛЯЕМЫХ КАК БЛОКИ!\nБудем делать"
            )
             files_list 'ru-yes 'ru-batch-normal-block-lib
          )
       );_ end of progn
      );_ end of if
   );_ end of progn
 );_ end of if
  (princ)
)
```

Главной здесь является функция `ru-batch-normal-block-lib` (листинг 24.6). Она демонстрирует типовой алгоритм, применяемый нами для обработки посторонних DWG-файлов:

1. Создается новый пустой *рабочий* файл.
2. В него вставляется *обрабатываемый* файл в виде блока.
3. Производится требуемая обработка.
4. Обработанный блок вновь сохраняется в файл[1].

В конкретных вариантах возможны нюансы (например, расчленение вставленного блока, запись в другой каталог, ведение LOG-файлов), но обязательным является дублированное несколько раз до и после обработки удаление "мусора", обеспечивающее постоянную "пустоту" *рабочего* файла.

Листинг 24.6. Функция `ru-batch-normal-block-lib`

```
(defun ru-batch-normal-block-lib
       (block_file / block_obj vla_sset_obj)
  (princ (strcat "\nОбрабатываю " block_file))
;;; Предварительная страховочная чистка мусора
  (vla-purgeall (ru-obj-get-active-document))
  (vla-purgeall (ru-obj-get-active-document))
  (vla-purgeall (ru-obj-get-active-document))
```

[1] Только не забудьте, что при этом будет сохранено только содержимое пространства модели обрабатываемого файла.

```
;;; Вставка обрабатываемого файла в виде блока
  (setq block_obj (ru-block-insert-obj block_file (list 0.0 0.0 0.0)
                    1 1 1 0))
;;; Основная обработка
  (ru-normal-all)
;;; Создание резервной копии
  (vl-file-copy block_file (ru-file-set-ext block_file ".bak"))
;;; Запись в файл
  (if (setq vla_sset_obj (ru-ss-entlast-to-vla-sset))
     (vla-wblock (ru-obj-get-active-document) block_file vla_sset_obj)
     (princ (strcat "\nОШИБКА. Не создан набор для записи " block_file))
 );_ end of if
;;; Удаление временного блока
  (ru-obj-delete-object block_obj)
  (vla-purgeall (ru-obj-get-active-document))
  (vla-purgeall (ru-obj-get-active-document))
  (vla-purgeall (ru-obj-get-active-document))
  (vla-purgeall (ru-obj-get-active-document))
  (vla-purgeall (ru-obj-get-active-document))
  (princ "\nготово\n")
)
```

Проследим, как выполняется обработка файла (листинги 24.7—24.10).

Листинг 24.7. Функция `ru-normal-all`

```
(defun ru-normal-all (/ ent)
  (princ "\nОбработка блоков...")
  (setq ent (tblnext "BLOCK" t))
  (while ent
    (ru-normal-block-props (cdr (assoc -2 ent)))
    (setq ent (tblnext "BLOCK"))
    (princ ".")
 );_ end of while
  (setq ent (entnext))
  (princ "\nОбработка примитивов...")
  (while ent
    (ru-normal-ent-props ent)
    (princ ".")
    (setq ent (entnext (cdr (assoc -1 (entget ent)))))
 );_ end of while
);_ end of defun
```

Листинг 24.8. Функция `ru-normal-ent-props`

```
(defun ru-normal-ent-props (ent)
;;; Нормализация примитивов
  (ru-ent-mod ent "0" 8)
  (ru-ent-mod ent "RU_CAD" 7)
  (ru-ent-mod ent "BYBLOCK" 62)
)
```

Листинг 24.9. Функция ru-normal-block-props

```
(defun ru-normal-block-props (ent)
;;; Нормализация вложенных примитивов блока
  (while ent
    (ru-normal-ent-props ent)
    (setq ent (entnext (cdr (assoc -1 (entget ent)))))
 );_ end of while
)
```

Листинг 24.10. Функция ru-ent-mod

```
(defun ru-ent-mod (ent value bit / ent_list old_dxf new_dxf)
;|Изменение свойства примитива ent, заданного DXF-кодом bit, на новое значение
value
Примеры:
 (ru-ent-mod (car (entsel)) 100 370)
 (ru-ent-mod (entlast) "1" 8)
|;
  (setq ent_list (entget ent)
        new_dxf  (cons bit
                       (if (and (= bit 62) (= (type value) 'str))
                         (if (= (strcase value) "BYLAYER")
                           256
                           0
                         );_ end of if
                         value
                       );_ end of if
                 );_ end of cons
 );_ end of setq
  (if (/= new_dxf (setq old_dxf (assoc bit ent_list)))
    (progn (entmod (if old_dxf
                     (subst new_dxf old_dxf ent_list)
                     (append ent_list (list new_dxf))
                   );_ end of if
           );_ end of entmod
           (entupd ent)
           (redraw ent)
   );_ end of progn
 );_ end of if
  ent
)
```

Используя приведенные функции в качестве прототипов, можно разработать программы для сколь угодно сложной обработки файлов.

Объектный доступ к другому документу

Мы рассмотрели пример с обработкой файлов, загружаемых в виде блоков. Этот способ имеет ряд недостатков, но главное преимущество в том, что все манипуля-

ции мы производим из своей, уже загруженной программы, т. к. мы продолжаем работать в активном документе. Возможен и вариант с "настоящей" загрузкой файлов в другие окна (если установлен многодокументный режим работы). В этом случае наша основная программа продолжает работать в пространстве имен прежнего "основного" документа, но может обрабатывать и другие документы объектными методами.

Обычно мы должны другой файл загрузить в отдельное окно, как-то обработать, а затем сохранить и закрыть. Напишем функцию открытия файла (листинг 24.11), возвращающую VLA-объект открытого документа или `nil` при неудаче.

Листинг 24.11. Функция `ru-dwg-open-other`

```
(defun ru-dwg-open-other (dwg_name)
  (ru-error-catch
    (function (lambda ()
                (vla-open (ru-obj-get-docs-collection) dwg_name)
              );_ end of lambda
   );_ end of function
    (function
      (lambda (x)
        (princ
          (strcat "\nОШИБКА RU-DWG-OPEN-OTHER: Нельзя открыть файл "
          dwg_name)
       );_ end of princ
        nil
     );_ end of lambda
   );_ end of function
 );_ end of ru-error-catch
);_ end of defun
```

Сохранить другой документ можно с помощью функции, приведенной в листинге 24.12.

Листинг 24.12. Функция `ru-dwg-save-other`

```
(defun ru-dwg-save-other (dwg_obj dwg_name)
  (ru-error-catch
    (function (lambda ()(vla-save dwg_obj)))
    (function
      (lambda (x)
        (princ
          (strcat "\nОШИБКА RU-DWG-SAVE-OTHER: Нельзя сохранить файл "
           dwg_name)
       );_ end of princ
        nil
     );_ end of lambda
   );_ end of function
 );_ end of ru-error-catch
);_ end of defun
```

И наконец, обработать другой документ можно с помощью функции, приведенной в листинге 24.13.

Листинг 24.13. Функция `ru-dwg-modify`

```
(defun ru-dwg-modify (dwg_file expresion / other_dwg_obj)
;; Открыли другой документ
  (if (setq other_dwg_obj (ru-dwg-open-other dwg_file))
    (progn
;|-------------------------------------------------------------------------
Что-то делаем. Для примера выведем имя документа, которое будет напечатано в окне
основного документа, а не того, который открыли
-------------------------------------------------------------------------|;
      (princ (strcat "\nОткрыт " (vla-get-fullname other_dwg_obj) "\n"))
;|-------------------------------------------------------------------------
Во вновь открытом документе мы можем выполнить только то, что доступно через
объектную модель. Мы предусматриваем отправку в командную строку любого выражения.
Выполняться оно будет в "том" документе.
-------------------------------------------------------------------------|;
      (vla-sendcommand other_dwg_obj expresion)
;|-------------------------------------------------------------------------
После выполнения действий сохраняем и закрываем документ
-------------------------------------------------------------------------|;
      (if (vla-get-saved other_dwg_obj)
        (ru-dwg-save-other other_dwg_obj dwg)
     );_ end of if
      (vla-close other_dwg_obj t dwg)
   );_ end of progn
 );_ end of if
);_ end of defun
```

Эта функция не претендует на совершенство и разработана в демонстрационных целях. Мы показали, что можно сделать и универсальную обработку — достаточно правильно "сконструировать" строку, передаваемую посредством метода `SendCommand`. В принципе, мы можем выполнить только такие же операции, как и из программ, написанных на Delphi или Visual Basic, но при этом имеем существенные преимущества:

- в нашем распоряжении все удобства обработки данных в LISP;
- в открытый документ уже загружена наша главная библиотека и в нем доступны все его функции.

Не надо забывать, что функции, определенные в файле программы (а не в автоматически загружаемых библиотеках), в "том" документе недоступны. Одно из преимуществ компоновки даже редко используемых функций в главную библиотеку и заключается в их доступности во всех открываемых документах. При необходимости мы можем загрузить и любую программу.

Более на этой методике мы задерживаться не будем, но читатели могут по этой схеме разрабатывать собственные, сколь угодно сложные программы.

Конвертирование файлов

Рассмотрим интересующий многих процесс *перекодировки* файлов. О проблемах, методиках и мифах, связанных с конвертированием файлов, мы подробно писали в *главе 4*, сейчас попытаемся разработать программу перекодировки. Всего лишь попытаемся, потому что абсолютно надежных программ такого типа мы пока не встречали — каждая подобная программа, в том числе и наши прежние варианты, может обработать тысячи своих файлов, но сломаться на первом же постороннем файле, в котором имеются какие-то нюансы.

Замечание

Попытавшись загрузить один из "древних" файлов в формате AutoCAD R12, мы получили сообщения об ошибке в какой-то строке какого-то CPP-файла, и прервать зацикленный вывод сумели только "варварским убийством" системы AutoCAD. Несколько раз получали и "любимое" сообщение *error: Exception occurred: 0xC0000005 (Access Violation)*. Так что учесть всю "чертовщину", которая может встретиться в файлах, невозможно.

А искали мы хоть какой-то файл, в котором были бы и тексты, и именованные объекты, и символы UNICODE.

Начнем разработку с выделения и решения частных задач. Прежде всего надо уметь конвертировать отдельные символы, затем строки, далее разыскивать в файле все строки, включая все, где могут встретиться "буквы". Неплохо уметь конвертировать символы из DOS в WIN и обратно (иногда и это требуется, но для экономии места листинги не будем приводить), а также преобразовывать строки, которые система AutoCAD без спросу перевела в UNICODE — они выглядят в виде символов косой черты с цифрами. Такие функции приведены в листингах 24.14 и 24.15.

Листинг 24.14. Функция `ru-conv-oem-char`

```
(defun ru-conv-oem-char (ch / code)
  (setq code (ascii ch))
  (chr (+ code
          (cond ((<= 128 code 175) 64)
                ((<= 176 code 223) -48)
                ((<= 224 code 239) 16)
                (T 0)
          );_ end of cond
       );_ end of +
 );_ end of chr
)
```

Листинг 24.15. Функция `ru-conv-oem-to-ansi`

```
(defun ru-conv-oem-to-ansi (str / st)
;| Конвертирование строки. Пример:
(ru-conv-oem-to-ansi
", з й е оJ  ¦E"-Ўл" жЕвагб, ¤, -® д "миЕЎл© нЄ§Ґ¬Ї"па")
"В чащах юга жил-был цитрус, да, но фальшивый экземпляр"
```

```
ОСТОРОЖНО! Повторная перекодировка портит строку! Пример:
(ru-conv-oem-to-ansi
  "В чащах юга жил-был цитрус, да, но фальшивый экземпляр")
"ґ чрщрх юур цшы-сыы цштрус, фр, эю фрыьшштыщ эъучхьяыяр"
|;
  (setq st "")
  (while (< 0 (strlen str))
    (setq
      st  (strcat st (ru-conv-oem-char (substr str 1 1)))
      str (substr str 2)
   );_ end of setq
 );_ end of while
  st
);_ end of defun
```

Умея конвертировать строки, мы, вроде бы, можем и переименовать все, что требуется. Однако, все не так просто. Попробуйте хотя бы переименовать командой RENAME (НОВОЕИМЯ) различные объекты (рис. 24.2). Команда "жалуется" на неверное *старое* имя! Да про *старое* имя мы знаем, проверять нужно *новое* имя!!! К тому же "неправильные" имена отображаются во всех диалоговых окнах системы AutoCAD, а список их выводится нашей функцией:

```
(ru-obj-list-layers) возвращает
("0" "ў...Ќ" "ў...Љ‹" "ЉЂЋЌ" "ЉЋЋ‚" "-ЋЉ" "„ЋЋ" "ЏЋщ'" "ЋЃЋ„" "'ў_Ѓ" "'ЋЋ‚"
 "Ќ...ЃЋ" "„...Ћ..." "ЋЋЊЉ" "-‚...’›" "'ў_Љ" "ASHADE" "‹€ў'")
```

Кстати, переведем их ради любопытства:

```
(mapcar '(lambda (name) (ru-conv-oem-to-ansi name)) (ru-obj-list-layers))
("0" "СТЕН" "СТЕКЛ" "КАРН" "КРОВ" "ЦОК" "ДОР" "ПОЯС" "ОБОД" "СТ_Б" "ТРАВ" "НЕБО"
"ДЕРЕ" "РАМА" "ЦВЕТЫ" "СТ_К" "ASHADE" "ЛИСТ")
```

Слои явно создавал русский архитектор, но даже он имел полное право, работая в системе AutoCAD R12, создавать такие имена.

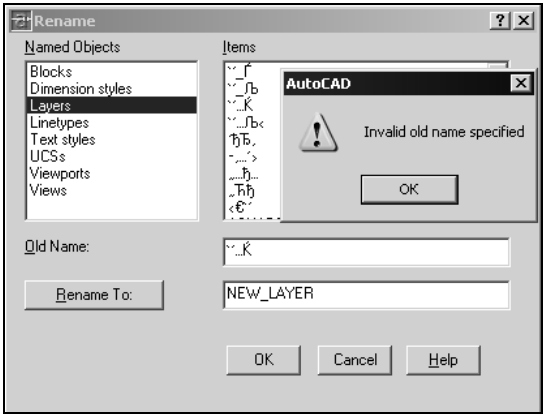


Рис. 24.2. Попытка переименования слоя

Подобная ситуация возникает со всеми именованными объектами, причем некоторые из них позволяется переименовывать, а некоторые — нет. Проверим старые имена функцией `snvalid`:

```
(mapcar '(lambda (name) (snvalid name)) (ru-obj-list-layers))
(T nil nil T nil nil nil nil nil nil nil nil T nil nil T nil)
```

Вот где причина беды — некто в фирме Autodesk не придумал ничего лучшего, как проверять старые имена этой функцией. Может быть переименование можно выполнить объектными методами? Что же, попробуем. Напишем функцию, изменяющую имя VLA-объекта (листинг 24.16). Встроенную функцию мы по традиции "обертываем" в свой обработчик ошибок, чтобы не происходила "ломка" основной программы из-за исключительных ситуаций.

Листинг 24.16. Функция `ru-obj-put-name`

```
(defun ru-obj-put-name (obj name)
  (ru-error-catch
    (function(lambda () (vla-put-name obj name) name))
    (function(lambda (x) (princ (strcat
      "\nОШИБКА RU-PUT-NAME: Нельзя присвоить имя " name)) NIL))
 )
)
```

Переименование слоя будем выполнять функцией, приведенной в листинге 24.17.

Листинг 24.17. Функция `ru-layer-obj-rename`

```
(defun ru-layer-obj-rename (old_name new_name / obj)
;;; Переименование слоя объектным методом
  (if (setq obj (ru-obj-layer-by-name old_name))
    (ru-obj-put-name obj new_name)
    (progn
      (princ (strcat "\nОШИБКА RU-LAYER-RENAME: Нет слоя " old_name))
      nil
   );_ end of progn
 );_ end of if
)
```

Проверка на практике:

```
(mapcar '(lambda (name) (ru-layer-obj-rename name
    (ru-conv-oem-to-ansi name))) (ru-obj-list-layers))
```

вернет

```
("0" nil nil "КАРН" nil nil nil nil nil nil nil nil nil "РАМА" nil nil "ASHADE" nil)
```

Ошибка кажется непреодолимой. Кстати, текстовый стиль объектным методом вообще нельзя переименовать, т. к. свойство `Name` для объекта `TextStyle` — read-only.

Ну, тупые!

Вы подумали, что эту фразу известного писателя-юмориста мы в очередной раз решили процитировать по поводу разработчиков фирмы Autodesk? Нет, это про нас сказано!

> Конкретно про меня. *(Сергей Зуев)*

Тысячи раз мы конвертировали файлы, разрабатывали много вариантов программ и ни разу не удосужились просто войти в открытую дверь, а пробивали стены лбом.

Цитируем высказывание из *главы 4*:

"Глупая в отношении всего "неамериканского" система Win-AutoCAD, загрузив файл с DWGCODEPAGE = dos866, радостно начинает конвертировать имеющиеся в ней тексты. Но делать она этого не умеет, и в результате вместо русских символов появляются символы косой черты с цифрами, соответствующими номеру символа в кодовой таблице".

Примите наши уверения — все происходило именно так. Далее мы советовали:

"Если предстоит конвертировать файлы системы AutoCAD R12, то необходимо избавиться от упоминания в них о DWGCODEPAGE = dos866. Сделать это можно, открыв рисунок в AutoCAD R12 и установив значение системной переменной DWGCODEPAGE, равное "undefined" (именно так!). Делать это много раз неудобно, да и система AutoCAD R12 может быть недоступной, поэтому лучше воспользоваться утилитой wnewcp.exe, которая делает это без запуска системы AutoCAD".

Действительно, после этого файлы становились обрабатываемыми, но возникали проблемы, с которыми мы и разбираемся.

А теперь приводим новый вариант совета по конвертированию файлов системы AutoCAD R12:

Если предстоит конвертировать файлы системы AutoCAD R12, то необходимо воспользоваться утилитой wnewcp.exe, и, с ее помощью, заменить кодовую страницу в файле AutoCAD R12 на DOS866 (рис. 24.3). Именно на DOS866, а не ANSI_1251_Cyrillic!

Обратите внимание на информацию о файле — указано **Code page:BIG5**. У нас давно нет системы AutoCAD R12, и мы не можем загрузить этот файл для проверки. Но, просматривая с помощью утилиты wnewcp.exe все файлы формата AC1009, мы не нашли ни одного, у которого кодовая страница распознавалась бы как dos_866 (за исключением тех, в которых кодовая страница была изменена самой утилитой wnewcp.exe) — везде или UNDEFINED, или BIG5. После конвертирования утилита wnewcp.exe выдает сообщение:

```
File: Plan_R12.DWG    Codepage changed  from BIG5 to DOS866.
```

Теперь мы просто загружаем "отконвертированный" файл в AutoCAD и видим, что все русские имена и надписи отображаются правильно в системе AutoCAD R14 и AutoCAD 2002[1]. Причем не только отображаются, изменены и сами надписи, в чем легко убедиться, загружая их в редактор текстовой строки. Система AutoCAD во время загрузки рисунка сообщила:

---

[1] На версиях 2000 и 2000i не проверяли, но предполагаем, что результат будет такой же.

```
Converting old drawing.
Opening a Release 12 format file.
Substituting [rucad.shx] for [AR].
Substituting [rucad.shx] for [CYRILSTD.SHX].
Substituting [rucad.shx] for [CYRILSTD.SHX].
Substituting [rucad.shx] for [CYRILSTD.shx].
Substituting [rucad.shx] for [CYRILSTD.shx].
Regenerating model.
```

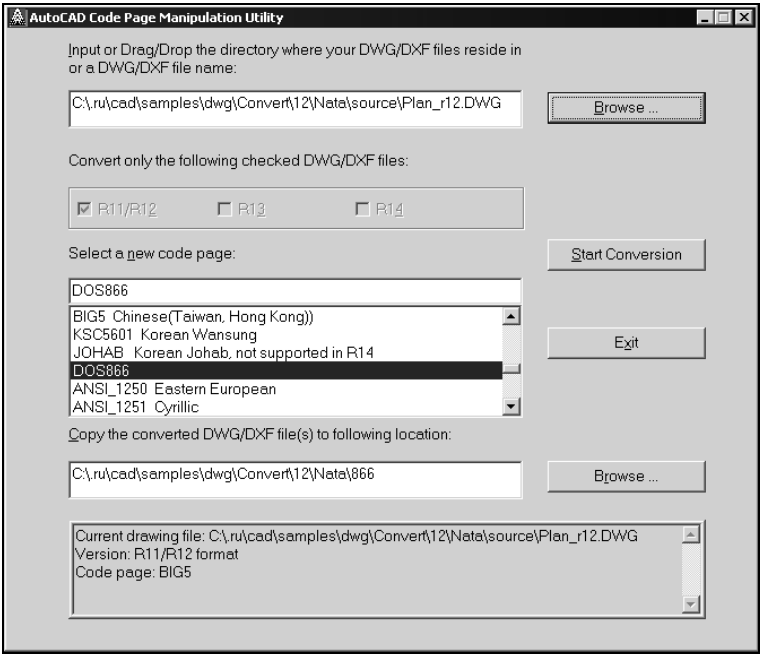


Рис. 24.3. Установка параметров для изменения кодовой страницы

На рис. 24.4 показаны два окна с разными версиями одного файла. Слева — файл, в котором системная переменная DWGCODEPAGE = dos866 установлена с помощью утилиты wnewcp.exe, справа — без обработки. Утилита wnewcp.exe рисунок не конвертирует, она изменяет всего два байта (рис. 24.5), значит система AutoCAD сумела правильно отконвертировать документ. Собственно, удивительно не это — так и должно быть, странно, почему она ранее этого не умела, и почему версия системы AutoCAD R12 неправильно записывала кодовую страницу.

Но система AutoCAD 2004 снова не умеет правильно отконвертировать русские символы! Появилась уже и версия AutoCAD 2005. Так что вопрос, кто "они", еще остается открытым. Впрочем, за истечением срока давности мы не будем выяснять, кто виноват, нам надо решить, что делать с файлами, которые по каким-то причинам не могут быть конвертированы автоматически.

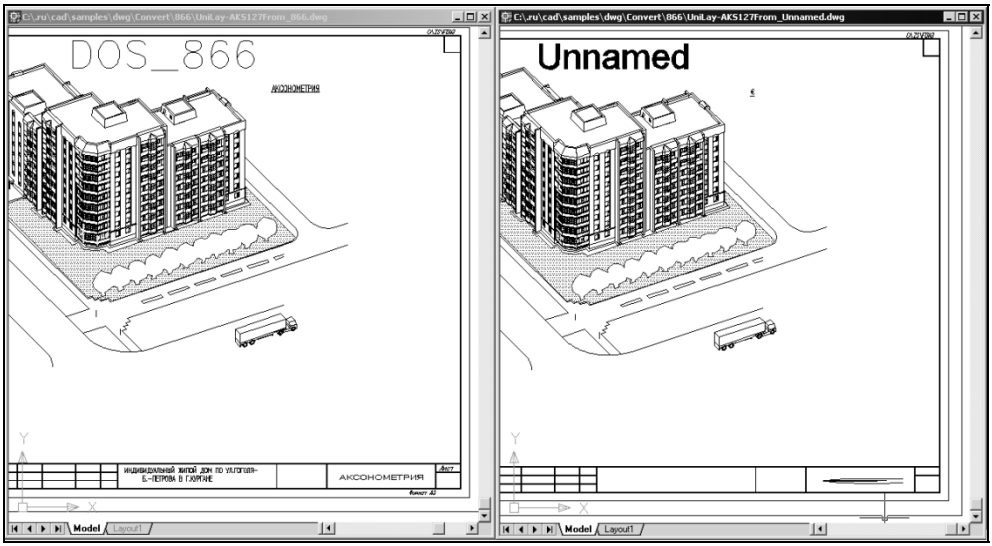


Рис. 24.4. Две версии одного файла

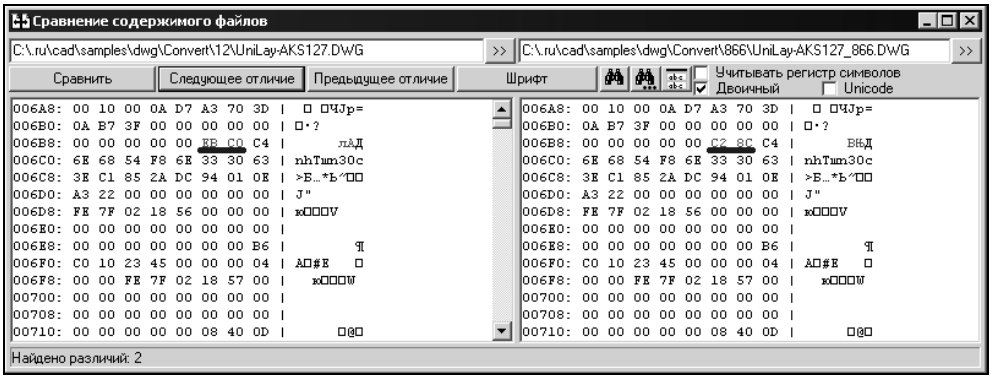


Рис. 24.5. Сравнение файлов до и после обработки утилитой wnewcp.exe. Подчеркнуты измененные байты

Как избежать переименования

Возможен такой прием — создавать новые именованные объекты с правильными именами с последующим изменением свойств всех примитивов. А старые объекты, после того как с ними не будут связаны примитивы, попытаемся удалить.

Такую схему мы пытались усовершенствовать несколько раз, но надежного решения не получалось — все время находились какие-то варианты, связанные с особенностями свойств разных примитивов. В результате мы решили принципиально изменить подход — поручить системе AutoCAD самой правильно переименовать именованные объекты, а уж с примитивами мы как-нибудь справимся.

При выполнении команды AUDIT (ПРОВЕРИТЬ) система AutoCAD заменяет неправильные имена (а с ее точки зрения это все русские слова в DOS-кодировке) на автоматически генерируемые типа `AUDIT_I_040302232920-32`. Мы можем проделать следующие "махинации":

1. Сохранить списки имен всех именованных объектов в DOS-кодировке.
2. Перекодировать их, чтобы знать, какими должны быть новые имена, но пока ничего не переименовывать.
3. Произвести аудит рисунка с заменой имен на правильные, но непонятные.
4. Произвести замену "аудитированных" имен на требуемые — при этом автоматически изменятся и свойства примитивов, такие как имена слоев.
5. Отконвертировать строковые значения у примитивов — к этому нет никаких препятствий.

Приступаем к реализации идеи. Как всегда, нам понадобятся вспомогательные функции, которые мы приведем после рассмотрения основной (листинг 24.18). Функции `ru-conv-file` передается один аргумент — имя функции, выполняющей конвертирование строк. Обычно это будет функция `ru-conv-oem-to-ansi`, но могут быть и другие варианты. Например, для передачи иностранным партнерам может потребоваться транслитерация латинскими символами.

Замечание

На LISP несложно разработать и функцию настоящего перевода (не символов, а слов и фраз) технических текстов, встречающихся в чертежах.

Листинг 24.18. Функция `ru-conv-file`

```
(defun ru-conv-file (quoted_func / audit_name audit_names_list blk
collection collection_name ename ent i j list_audit_names list_new_names
name names names_list new_names_list obj old_names_list sub_list _ent_modify)
;|-----------------------------------------------------------------------
Локальная функция модификации списка данных примитива. Модифицирует строковые
значения из списка ent, заданные DXF-кодами, с помощью функции quoted_func.
Возвращает измененный список данных примитива.
-----------------------------------------------------------------------|;
  (defun _ent_modify (ent dxf_code quoted_func is_print / old_value new_value)
    (if (assoc dxf_code ent)
      (progn
        (setq old_value (vl-princ-to-string (cdr (assoc dxf_code ent))))
        (if (= (type quoted_func) 'STR)
          (setq new_value quoted_func)
          (setq new_value ((eval quoted_func) old_value))
       );_ end of if
;; Ограничиваем явно бессмысленные действия
        (if (and (/= old_value new_value)
                 (/= old_value "DEFPOINTS")
                 (/= old_value "0")
          );_ end of and
```

```
          (progn
            (if (and is_print *ru_msg_debug*)
             (princ (strcat "\nИзменяю значение для DXF-кода "
             (itoa dxf_code) " [" old_value "] на [" new_value "]\n"))
           );_ end of if
            (setq ent (subst
                        (cons dxf_code new_value)
                        (cons dxf_code old_value)
                        ent
                      );_ end of subst
           );_ end of setq
         );_ end of progn
       );_ end of if
     );_ end of progn
   );_ end of if
    ent
 );_ end of defun
;|-----------------------------------------------------------------------
Локальная функция формирования списка имен именованных объектов из всех семейств,
допускающих переименование. В список включаем имена семейств
-----------------------------------------------------------------------|;
  (defun _make-names-list ()
    (list (list "Layers" (ru-obj-list-layers))
          (list "Dimstyles" (ru-obj-list-dim-styles))
          (list "Textstyles" (ru-obj-list-textstyles))
          (list "Layouts" (ru-obj-list-layouts))
          (list "Blocks" (ru-obj-list-blocks))
          (list "Views" (ru-obj-list-views))
          (list "Viewports" (ru-obj-list-viewports))
          (list "Linetypes"
                (ru-obj-list-collection-member-names
                  (ru-obj-doc-collection "Linetypes")
                );_ end of ru-obj-list-collection-member-names
        );_ end of list
          (list "UserCoordinateSystems"
                (ru-obj-list-collection-member-names
                  (ru-obj-doc-collection "UserCoordinateSystems")
                );_ end of ru-obj-list-collection-member-names
        );_ end of ru-obj-list-collection-member-names
   );_ end of list
 );_ end of defun

;;;--------Главная функция -------------------------------------
;|-----------------------------------------------------------------------
Составляем список имен именованных объектов
-----------------------------------------------------------------------|;
(setq old_names_list (_make-names-list))
;|-----------------------------------------------------------------------
```

Получили примерно такой список, в данном примере с русскими именами в DOS-кодировке (хотя это могли бы быть и имена в WIN-кодировке):

```
(
  ("Layers" ("0" ""ЂђЂЂ'" "ЃђЂЌЄ->" "DEFPOINTS" "'‘...Ќ›"))
  ("Dimstyles" ("Standard"))
  ("Textstyles" ("'‘ЂЌ„Ђђ'" "STANDARD" "ICAD"))
  ("Layouts" ("Layout1" "Model"))
  ("Blocks" ("*Model_Space" "*Paper_Space" "RECTANG" "$ZN71" "_KVADRAT"
             "*X13" "*X14" "_LINE15000Ђ")
 )
  ("Views" nil)
  ("Viewports" ("*Active"))
  ("Linetypes" ("ByBlock" "ByLayer" "CONTINUOUS" "CENTER"))
  ("UserCoordinateSystems" nil)
)
(cdr (list "Textstyles" (list "'‘ЂЌ„Ђђ'" "STANDARD" "ICAD")))
Теперь формируем другой список с этими же именами, но перекодированными функцией
----------------------------------------------------------------------|;
 (setq new_names_list
  (mapcar
   (function
      (lambda (names_list)
          (list (car names_list)
              (mapcar (function (lambda (name)
                                  (if name (eval quoted_func) name) name))
                      );_ end of function
                      (cadr names_list)
               );_ end of mapcar
             );_ end of list
           );_ end of lambda
         );_ end of function
       old_names_list
    );_ end of mapcar
 );_ end of setq
;|----------------------------------------------------------------------
Список "переводов" выглядит примерно так:
(
 ("Layers" ("0" "ФОРМАТ" "ГРАНИЦЫ" "DEFPOINTS" "СТЕНЫ"))
  ("Dimstyles" ("Standard"))
  ("Textstyles" ("СТАНДАРТ" "STANDARD" "ICAD"))
  ("Layouts" ("Layout1" "Model"))
  ("Blocks" ("*Model_Space" "*Paper_Space" "RECTANG" "$ZN71" "_KVADRAT"
       "*X13" "*X14" "_LINE15000M"))
  ("Views" nil)
  ("Viewports" ("*Active"))
  ("Linetypes" ("ByBlock" "ByLayer" "CONTINUOUS" "CENTER"))
  ("UserCoordinateSystems" nil)
)
Выполняем проверку с исправлением ошибок
----------------------------------------------------------------------|;
  (vla-auditinfo (ru-obj-get-active-document) :vlax-true)
```

```
;|-------------------------------------------------------------------------
Формируем список исправленных имен.
------------------------------------------------------------------------|;
  (setq audit_names_list (_make-names-list))
;|-------------------------------------------------------------------------
Такой же список, как до аудита, но с исправленными именами
(("Layers" ("0" "AUDIT_I_040302232920-32" "AUDIT_I_040302232920-33" "DEFPOINTS"
"AUDIT_I_040302232920-46"))
  ("Dimstyles" ("Standard"))
  ("Textstyles" ("AUDIT_I_040302232920-47" "STANDARD" "ICAD"))
  ("Layouts" ("Layout1" "Model"))
  ("Blocks" ("*Model_Space" "*Paper_Space" "RECTANG" "$ZN71" "_KVADRAT"
             "*X13" "*X14" "_LINE15000Ь"))
  ("Views" nil)
  ("Viewports" ("*Active"))
  ("Linetypes" ("ByBlock" "ByLayer" "CONTINUOUS" "CENTER"))
  ("UserCoordinateSystems" nil)
)
Далее переходим к ответственной операции - обратному переименованию имен по списку
audit_names_list в имена по списку new_names_list
------------------------------------------------------------------------|;
  (setq i 0)
  (foreach collection_list audit_names_list
    (setq collection_name (car collection_list)
          list_audit_names
           (cadr collection_list)
          list_new_names
           (cadr (nth i new_names_list))
          i (1+ i)
          j 0
   );_ end of setq
    (if list_new_names
      (progn
        (setq collection (ru-obj-doc-collection collection_name))
        (foreach name list_new_names
          (setq audit_name
                 (nth j list_audit_names)
                j (1+ j)
         );_ end of setq
          (if *ru_msg_debug*
            (princ (strcat "\n" audit_name "=" name))
         );_ end of if
;|-------------------------------------------------------------------
 Для текстовых стилей у объектов свойство Name только для чтения, поэтому их можно
переименовать только командой
------------------------------------------------------------------------|;
          (if (= collection_name "Textstyles")
;|-------------------------------------------------------------------------
 Опасность! В стилях могут быть пустые имена, образующиеся при применении SHAPE!!
------------------------------------------------------------------------|;
```

```
            (if (and (/= audit_name "") (/= name ""))
              (command "_.-RENAME" "_Style" audit_name name)
            );_ end of if
             (if
               (setq obj
                  (ru-obj-collection-item-by-name collection_name
                     audit_name)
              );_ end of setq
                (if (/= audit_name name)
;;; Не переименовываем анонимные блоки, штриховки и т. п.
                  (if (/= (substr audit_name 1 1) "*")
;;; Переименование объектным методом
                    (ru-obj-put-name obj name)
                );_ end of if
              );_ end of if
          );_ end of if
        );_ end of if
      );_ end of foreach
    );_ end of progn
   );_ end of if
 );_ end of foreach
;|-------------------------------------------------------------------------
Самое сложное позади. Приступаем к обработке примитивов
-------------------------------------------------------------------------|;
  (princ "\nОбработка блоков...\n")    ;
  (setq blk (tblnext "BLOCK" T))       ;
  (while blk                           ;
;; А теперь беремся за примитивы, входящие в блок
    (setq ent (cdr (assoc -2 blk)))
    (while ent
      (setq ent (_ent_modify
                  (_ent_modify
                    (_ent_modify
                      (_ent_modify (entget ent) 1 quoted_func nil)
                      2
                      quoted_func
                      nil
                    );_ end of _ent_modify
                    3
                    quoted_func
                    nil
                  );_ end of _ent_modify
                  4
                  quoted_func
                  nil
                );_ end of _ent_modify
    );_ end of setq
      (entmod ent)
;; Исследуем вложенные примитивы
      (setq ent (entnext (cdr (assoc -1 ent))));
   );_ end of while
```

```
    (setq blk (tblnext "BLOCK"))
 );_ end of while
;;; Покончили с блоками, беремся за объекты, "плавающие" на поверхности
  (setq ent (entnext))
  (princ "\nОбработка примитивов...\n")
  (while ent
    (setq ent   (entget ent)
          ename (cdr (assoc 0 ent))
   );_ end of setq
    (if (/= ename "SEQEND")
      (progn
        (setq ent (_ent_modify ent 1 quoted_func T))
       ;; Текст
        (if (/= ename "INSERT")
;;; Опасно! Эта проверка нужна, иначе будет обратное переименование
          (setq ent (_ent_modify ent 2 quoted_func T))
       );_ end of if
        (setq ent (_ent_modify ent 3 quoted_func T))
        (setq ent (_ent_modify ent 4 quoted_func T))
        (entmod ent)
     );_ end of progn
   );_ end of if
    (setq ent (entnext (cdr (assoc -1 ent))))
 );_ end of while
  (command "_.REGEN")
  (princ "\n Готово\n")
  (princ)
);_ end of defun
```

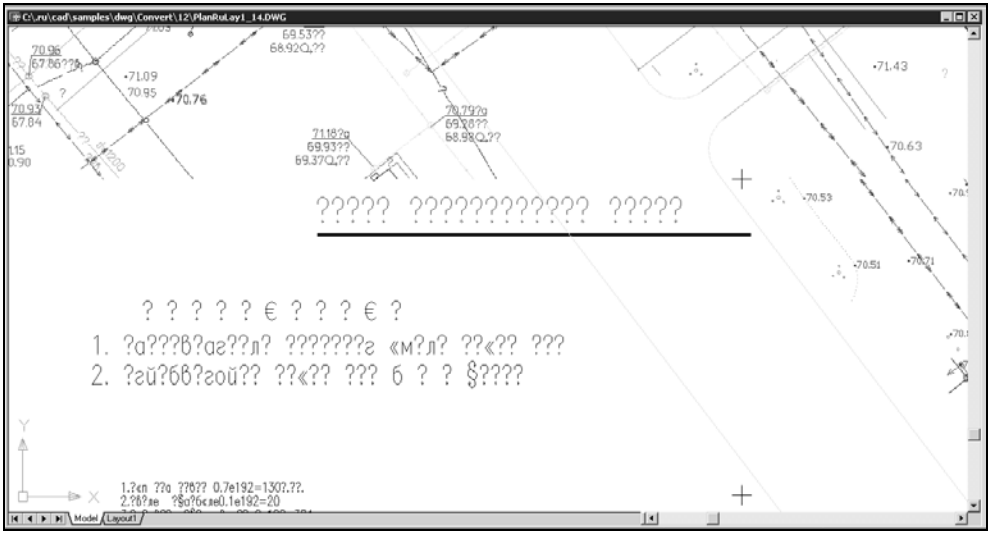


Рис. 24.6. Файл до конвертирования

Остается надеяться, что в AutoCAD 2005 не появятся новые "подлые ловушки", требующие очередной переработки функции.

Испытаем функцию в работе. Загрузим файл с DOS-кодировкой (рис. 24.6) и отконвертируем его: `(ru-conv-file 'ru-conv-oem-to-ansi)`.

Результат можно увидеть на рис. 24.7.

Рис. 24.7. Файл после конвертирования из кодировки DOS866 в WIN1251

Рис. 24.8. Файл после транслитерации

А теперь представьте, что заказчику требуются рисунки только с латинскими символами — "у богатых свои причуды" — может быть, у него русских шрифтов нет, но полно иммигрантов, понимающих русские слова. Имея нашу функцию, произвести транслитерацию очень просто: `(ru-conv-file 'ru-conv-string-win-translit)`. Результат транслитерации показан на рис. 24.8.

Функции конвертирования можно использовать и для массовой обработки файлов с применением рассмотренной ранее функции `ru-batch-file-operations`.

Приведение текстового стиля для всех примитивов к определению

Неожиданно популярной оказалась программа приведения всех надписей (тексты, атрибуты блоков) к виду, заданному текущим стилем. Проблема заключается в том, что надписи, написанные одним стилем, могут иметь разный угол наклона и ширину. При изменении определения стиля в диалоговом окне вид надписей не изменяется. Положение осложняется тем, что некоторые начальники любят отдавать приказы "чтобы все было курсивом" (или наоборот). Кроме того, в рисунок могут попасть блоки, созданные в разное время, под разные вкусы. Для быстрого решения такой проблемы и написана следующая программа. До ее запуска необходимо в диалоговом окне команды STYLE (СТИЛЬ) настроить требуемые параметры каждого стиля.

Программа должна запросить (рис. 24.9), какие стили приводить к определению и выполнить задуманное (листинг 24.19). Стиль RU_LINE, используемый нами для создания "лохматых" типов линий, не должен менять начертание и не может быть выбран в диалоговом окне.

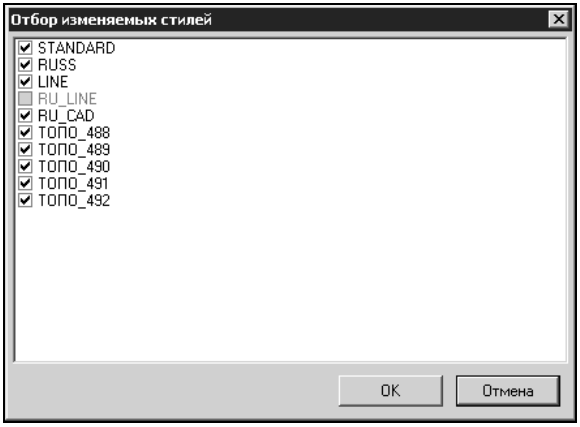


Рис. 24.9. Отбор текстовых стилей для приведения к определению

Листинг 24.19. Файл ru-txt_style_redraw.lsp

```
(defun START (/ checked_list enabled_list i res_list text_styles_list _update-style
_modify-style)
```

```
;; Локальная функция
  (defun _modify-style (style_name new_style_name ent_data style_width
            style_angle)
    (if (and
          (or
            (= "TEXT" (cdr (assoc 0 ent_data)))
            (= "ATTRIB" (cdr (assoc 0 ent_data)))
            (= "ATTDEF" (cdr (assoc 0 ent_data)))
        );_ end of or
         (or
           (= "*" style_name)
           (= (strcase style_name) (cdr (assoc 7 ent_data)))
        );_ end of or
       );_ end of and
      (setq ent_data (entmod
                       (subst
                         (cons 7 new_style_name)
                         (assoc 7 ent_data)
                         ent_data
                       );_ end of subst
                     );_ end of entmod
            ent_data (entmod
                       (subst
                         (cons 41 style_width)
                         (assoc 41 ent_data)
                         ent_data
                       );_ end of subst
                     );_ end of entmod
            ent_data (entmod
                       (subst
                         (cons 51 style_angle)
                         (assoc 51 ent_data)
                         ent_data
                       );_ end of subst
                     );_ end of entmod
    );_ end of setq
   );_ end of if
 );_ end of defun
;; Локальная функция
  (defun _update-style (style / ent_name ent_data old_style style_angle
     style_width tbe)
 (princ (strcat "Ждите, привожу стиль\n '" style "' к определению...\n"))
 (setq ent_name    (tblnext "DIMSTYLE" t)
     old_style   (tblobjname "STYLE" style)
     tbe         (entget old_style)
     style_width (cdr (assoc 41 tbe))
     style_angle (cdr (assoc 50 tbe))
);_ end of setq
 (while ent_name
  (setq ent_data
    (entget (tblobjname "DIMSTYLE" (cdr (assoc 2 ent_name)))))
```

```
   (if (or
         (= "*" style)
         (= old_style (cdr (assoc 340 ent_data)))
       );_ end of or
      (entmod (subst (cons 340 old_style) (assoc 340 ent_data) ent_data))
  );_ end of if
   (entupd (cdr (assoc -1 ent_data)))
   (setq ent_name (tblnext "DIMSTYLE"))
);_ end of while
 (setq ent_name (tblnext "BLOCK" T))
 (while ent_name
   (setq ent_data (cdr (assoc -2 ent_name)))
   (while ent_data
     (setq ent_data (entget ent_data))
     (_modify-style style style ent_data style_width style_angle)
     (setq ent_data (entnext (cdr (assoc -1 ent_data))))
  );_ end of while
   (setq ent_name (tblnext "BLOCK"))
);_ end of while
;; примитивы
 (setq ent_name (entnext))
 (while ent_name
   (_modify-style style style (entget ent_name) style_width style_angle)
   (entupd ent_name)
   (setq ent_name (entnext ent_name))
);_ end of while
);_ end of defun
;;;-Главная функция -----------
  (if (ru-yes (strcat
    "\nПроизводится изменение начертания всех существующих "
   "\nТЕКСТОВ, АТРИБУТОВ \nво ВСЕХ примитивах, включая блоки,"
   " \nвыбранного стиля!"
   "\n\nУстанавливается по определению стиля"
   "\n начертание, сжатие, угол наклона"
   "\nПользоваться осторожно!\nБудем делать"
     )
  (progn
   (setq text_styles_list (ru-obj-list-textstyles))
   (foreach style text_styles_list
     (setq enabled_list (cons (if (= style "RU_LINE") nil T)
                              enabled_list);_ end of cons
           checked_list (cons (if (= style "RU_LINE") nil T)
                              checked_list
                        );_ end of cons
    );_ end of setq
  );_ end of foreach
   (if (setq res_list (ru-dlg-show-check-list "Отбор изменяемых стилей"
     text_styles_list (reverse checked_list) (reverse enabled_list))
      );_ end of setq
    (progn
       (setq i 0)
```

```
      (foreach style text_styles_list
         (if (nth i res_list) (_update-style style))
         (setq i (1+ i))
      );_ end of foreach
   );_ end of progn
  );_ end of if
 );_ end of progn
);_ end of if
(princ)
);_ end of defun
(START)
```

Используя приведенные функции для изменения свойств в качестве прототипов, можно разработать много специализированных программ.

Обработка штриховок

Опытные "автокадчики" стараются применять штриховки в минимальном объеме, но в некоторых разделах проектов без штриховок никак не обойтись. При проектировании по стандартам разновидностей образцов штриховок требуется немного. Беда в том, что многих именно таких, какие нужны, в системе AutoCAD нет. Разумеется, за годы работы с AutoCAD создано множество образцов и вряд ли стоит теперь заниматься разработкой собственных. Все можно "достать".

После того как нужные образцы добыты, ими нужно суметь с толком распорядиться. Коллекции штриховок обычно собирают в файлы acad.pat и acadiso.pat (иногда, в соответствии с рекомендациями фирмы Autodesk, для каждого образца создают собственный файл). В результате набирается множество файлов штриховок неизвестного происхождения, в каждом из которых есть и стандартные образцы, и собственные, и "левые". Одноименные образцы могут быть выполнены в разных единицах (английских или метрических). Как разобраться с этим "добром" (скорее, "барахлом")?

Для начала напишем функцию, раскладывающую PAT-файл на отдельные образцы (листинг 24.20)[1]. Ей передается имя файла с образцами и имя каталога, в который нужно помещать отдельные файлы. Функцией `ru-hatch-split-pat-file` можно обработать все имеющиеся PAT-файлы, складывая образцы в один каталог. Для одноименных образцов будут создаваться новые имена файлов путем конкатенации имени образца и уникального кода, например:

- ANSI37.pat — первый файл с образцом ANSI37;
- ANSI37_04030809550763.pat — второй файл с образцом ANSI37;
- ANSI37_04030809550942.pat — третий файл с образцом ANSI37.

Листинг 24.20. Функция `ru-hatch-split-pat-file`

```
(defun ru-hatch-split-pat-file (pat_file dir / count file_name new
                                pat_list pat all_pat_list short_pat)
```

[1] Это может быть и консольная программа, написанная на любимом языке программирования.

```
 (setq dir       (ru-dirs-path-and-slash dir)
       pat_list  '()
       short_pat (strcat (vl-filename-base pat_file) ".pat")
       pat       short_pat
       count     0
);_ end of setq
 (if (setq all_pat_list (ru-list-read-from-file pat_file))
   (progn
     (setq new nil)
     (foreach string all_pat_list
       (if (= (substr string 1 1) "*")
         (progn
           (if (> count 0)
             (progn
               (setq
                 pat_list (cons (strcat ";;") pat_list)
                 pat_list (cons (strcat ";; END OF " pat) pat_list)
                 pat_list (cons (strcat ";;") pat_list)
              );_ end of setq
              (print (strcat "Записываю " file_name))
              (ru-list-write-to-file file_name (reverse pat_list))
             );_ end of progn
           );_ end of if
           (setq new       T
                 pat       (ru-string-word string 1 '("*" ","))
                 pat_list '()
                 pat_list (cons (strcat "") pat_list) pat_list
                          (cons (strcat ";; FROM: " short_pat) pat_list)
                 pat_list
                          (cons (strcat ";; COMMENT: ") pat_list)
                 pat_list
                           (cons (strcat ";;") pat_list)
                 pat_list  (cons string pat_list)
                 new       nil
                 count     (1+ count)
                 file_name (strcat dir pat ".pat")
           );_ end of setq
           (print (strcat "Добавляю " string))
           (if (findfile file_name)
             (setq
               file_name
                (strcat dir pat "_" (ru-string-unique) ".pat")
              );_ end of setq
           );_ end of if
         );_ end of progn
         (if (/= (substr string 1 1) ";")
            (setq pat_list (cons string pat_list))
         );_ end of if
       );_ end of if
     );_ end of foreach
  );_ end of progn
```

```
    (ru-msg-alert (strcat "Не могу прочитать " pat_file))
 );_ end of if
  (princ)
);_ end of defun
```

Полученные образцы придется просмотреть и спрятать ненужные дубли в надежное место. Оставшиеся можно скомпоновать обратно в сборный PAT-файл с помощью функции `ru-hatch-merge-pat-files` (листинг 24.21).

Листинг 24.21. Функция `ru-hatch-merge-pat-files`

```
(defun ru-hatch-merge-pat-files (out_file pat_files_list /
                                 pat_list pat_string_list)
  (mapcar
    (function (lambda (pat_file)
                (setq pat_list        (ru-list-read-from-file pat_file)
                      pat_string_list (append pat_list pat_string_list)
                );_ end of setq
              );_ end of lambda
    );_ end of function
    (reverse pat_files_list)
 );_ end of mapcar
  (ru-list-write-to-file out_file pat_string_list)
);_ end of defun
```

Лучше всего собрать образцы в файл acadiso.pat в нашем каталоге `%LocalAcadAllVersionDir%`.

Теперь можно написать функцию для рисования сборника всех образцов штриховок (листинг 24.22).

Листинг 24.22. Функция `ru-hatch-draw-samples`

```
(defun ru-hatch-draw-samples (pat_list / def_scale dx dxcurr n row
  sample_height sample_width t0 tx x xcurr xmax y ycurr)
;;;Рисование матрицы образцов штриховок

  (defun draw_sample (pt0 x y pattern / pt2 pt3 pt4 ss pttext)
    (setq
      pt2    (polar pt0 0 x)
      pt4    (polar pt0 (ru-geom-go-left 0) y)
      pt3    (polar pt2 (ru-geom-go-left 0) y)
      pttext (polar pt4 (ru-geom-go-left 0) (* y 0.1))
   );_ end of setq
    (ru-pline-add (list pt0 pt2 pt3 pt4) T 0 0 nil)
    (setq ss (ssadd) ss (ssadd (entlast) ss))
    (ru-hatch-add-obj pattern (ru-normal-hatch-scale pattern) 0 ss)
    (ru-text-add pattern pttext (* y 0.15) 0 "L")
 );_ end of defun
```

```
(setq sample_width  45.0
      sample_height 30.0
     t0 (ru-get-point-reguired "Левый верхний угол матрицы образцов" nil)
     dx (ru-get-dist "Расстояние между образцами " (* sample_width 2) t0)
     xmax (ru-get-dist "\nРазмер матрицы максимальный: " (* dx 20) t0)
     tx t0 n 0 row 0
 );_ end of setq
(foreach name pat_list
 (setq dxcurr (* dx n) xcurr  (+ (car t0) dxcurr) ycurr  (cadr tx))
 (if (> dxcurr xmax)
   (setq n 0 row (1+ row) xcurr (car t0) ycurr (- (cadr t0) (* dx row)))
);_ end of if
 (setq tx (list xcurr ycurr) n  (1+ n))
 (draw_sample tx sample_width sample_height name)
);_ end of foreach
 (print "Готово.")
 (princ)
)
```

А как получить список образцов штриховок, известных системе AutoCAD? Сама-то она знает об их существовании, раз выводит в диалоговом окне команды BHATCH (КШТРИХ) список, да еще умудряется генерировать миниатюры изображений. К сожалению, мы не нашли ответа на этот вопрос. Не существует некоей таблицы штриховок, наподобие таблицы блоков. Используя пару функций (листинги 24.23 и 24.24) можно получить список образцов *нарисованных* штриховок, но для получения списка доступных образцов мы не придумали ничего лучшего, чем чтение PAT-файла (листинг 24.25).

Листинг 24.23. Функция `ru-obj-list`

```
(defun ru-obj-list (obj_name / item result)
 (vlax-for item (ru-obj-get-model-space)
   (if (= (vlax-get-property item 'ObjectName) obj_name)
            (setq result (append (list item) result))))
  result
); defun
```

Листинг 24.24. Функция `ru-obj-list-hatch-patterns`

```
(defun ru-obj-list-hatch-patterns (/ hatch result)
  (foreach hatch (ru-obj-list "AcDbHatch")
    (setq result
           (append (list (vlax-get-property hatch 'PatternName)) result)
   );_ end of setq
 );_ end of foreach
  ;_ end of vlax-for
  result
)
```

Листинг 24.25. Функция `ru-hatch-list-pat-in-file`

```
(defun ru-hatch-list-pat-in-file (pat_file / pat pat_list)
(foreach string (ru-list-read-from-file pat_file)
  (if (= (substr string 1 1) "*")
      (setq pat (ru-string-word string 1 '("*" ","))
        pat_list (cons pat pat_list))))
 (reverse pat_list)
)
```

Настройка оптимальных масштабов штриховок

Создадим новый файл с единицами рисунка мм и масштабом 1:1 и выполним функцию `ru-hatch-draw-samples`.

В результате у нас должен получиться сборник образцов штриховок наподобие показанного на рис. 24.10. Должен, но наверняка не получится.

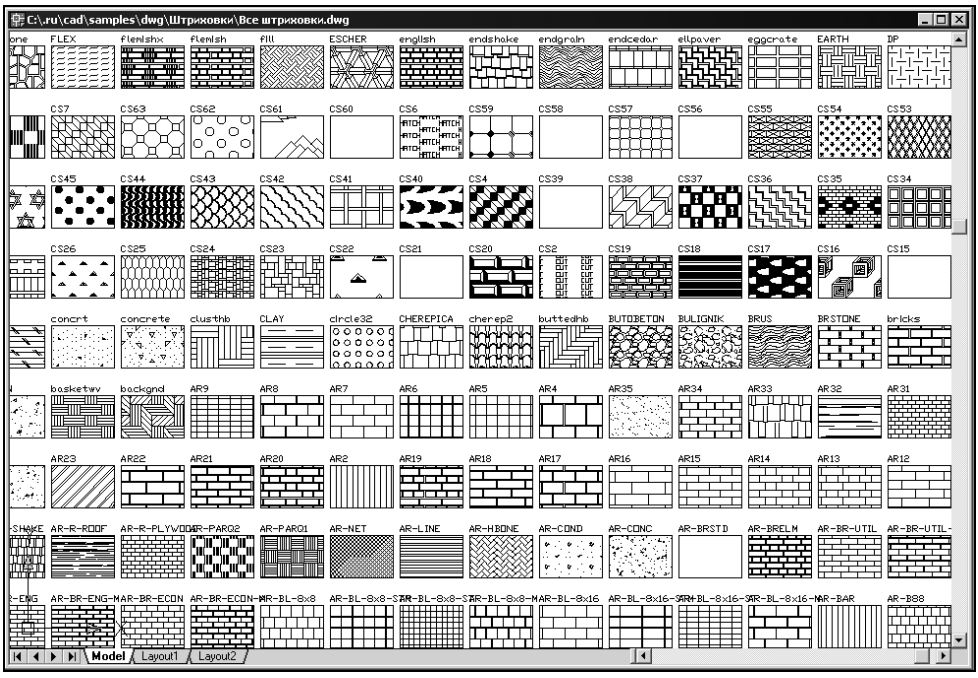


Рис. 24.10. Сборник образцов штриховок

Штриховки в PAT-файле могут быть созданы разными людьми и рассчитаны на разные системы измерений. Хотя мы создавали размеры образцов штриховок 45×30 мм на бумаге (оптимальный для восприятия), не все виды штриховок смогут правильно отобразиться в таком поле — некоторые будут слишком густыми, некоторые просто не впишутся. Наша функция рисования штриховок `ru-hatch-add-obj`

разработана так, что при ошибке рисования программа не прерывается, а выводится сообщение. В результате мы можем получить примерно такой протокол:

```
Сбой RU-HATCH-ADD-OBJ: Pattern cutstone, ОШИБКА: Automation Error. Hatch pattern
too dense
Сбой RU-HATCH-ADD-OBJ: Pattern CS60, ОШИБКА: Automation Error. Ambiguous output
Сбой RU-HATCH-ADD-OBJ: Pattern CS31, ОШИБКА: Automation Error. Invalid input
Сбой RU-HATCH-ADD-OBJ: Pattern concrt, ОШИБКА: Automation Error. Hatch pattern too
dense
```

Некоторые ошибки возникают из-за того, что по умолчанию система AutoCAD не может создать образец из более чем 10 000 сегментов. В этом случае обычно генерируется знакомое всем сообщение "`Spacing too dense or hatch pattern too small`". Для решения этой проблемы, конечно, нужно увеличивать масштаб штриховки, но в нашем случае, когда образцов много, а масштаб пока неизвестен, можно изменить настройки системы AutoCAD. Для этого достаточно в командной строке выполнить выражение `(setenv "MaxHatch" "100000")` и получить возможность создавать образец, состоящий от 100 до 10 000 000 сегментов. При этом значительно возрастет потребность в памяти, и производительность будет резко снижена.

После нескольких экспериментов мы получим набор из большинства образцов. Остальные, у которых прямоугольники остались пустые, но есть надписи с названиями образца, можно заштриховать командой ВНАТСН (КШТРИХ), подбирая масштаб вручную. На следующем этапе мы подберем оптимальные масштабы штриховок — такие, чтобы образец выглядел примерно так, как он должен выглядеть на бумаге. Для этого напишем функцию подбора масштаба (листинг 24.26).

Листинг 24.26. Функция `ru-hatch-change-scale`

```
(defun ru-hatch-change-scale (ename / ent pattern  ms new_scale old_scale
     do)
;;; изменение масштаба примитива-штриховки с записью в INI-файл
  (setq ent (entget ename) do t old_scale (cdr (assoc 41 ent))
        pattern (cdr (assoc 2 ent)) new_scale old_scale)
  (while do
    (setq ent (entget ename) old_scale (cdr (assoc 41 ent))
     new_scale (_ru-get-with-default "Масштаб штриховки"
                 (rtos old_scale 2 2) 'getreal nil "+ - Save Exit" nil)
   );_ end of setq
    (cond
      ((= new_scale "+") (setq old_scale (* old_scale 0.5)))
      ((= new_scale "-") (setq old_scale (* old_scale 2.0)))
      ((= new_scale "Save")
       (setq
         ms (rtos (ru-conv-unit-to-millimeter-in-paper old_scale) 2 3)
      );_ end of setq
       (ru-ini-write (ru-file-ini "hatch.ini") "HatchScale" pattern ms)
       (princ (strcat "\nШтриховка " pattern
          " описана в HATCH.INI с масштабом " ms "\n"))
       (setq do nil)
     )
```

```
      ((= new_scale "Exit") (setq do nil))
      (T (setq old_scale new_scale))
   );_ end of cond
    (if do (command "_.-HATCHedit" ename "_P" "" old_scale ""))
 );_ end of while
  old_scale
)
```

Теперь можно написать программу, позволяющую установить и сохранить оптимальный масштаб штриховки (листинг 24.27).

Листинг 24.27. Файл ru_set_hatch_best_scale.lsp

```
(defun START (/ ent)
  (ru-app-begin)
  (while (setq ent (ru-get-entsel-by-type "Выбери штриховку"
      "Это не ШТРИХОВКА" (list "HATCH") T))
      (ru-hatch-change-scale (car ent))
 )
  (ru-app-end)
  (princ)
)
(START)
```

Поработав с нарисованными образцами, мы запомним оптимальные, с нашей точки зрения, масштабы, которые и будут использоваться по умолчанию функциями рисования штриховок *(см. главу 25)*.

Обработка блоков

Мы уже рассмотрели "нормализацию" библиотечных блоков, но нам необходимо автоматизировать еще несколько рутинных операций, связанных с блоками. Для формирования библиотек блоков потребуются две простых программы — выгрузки всех блоков из активного рисунка в отдельные файлы (листинг 24.28) и вставка всех отобранных файлов (листинг 24.29).

Листинг 24.28. Файл ru_block_save_file.lsp

```
(defun START (/ descr file folder lst_names_descr name)
;;;извлечение блоков из чертежа
(if (ru-user-action-enabled "DEVELOP" T)
 (if (ru-yes (strcat "Программа выгружает на диск"
                "\n ВСЕ БЛОКИ,\nЗАМЕНЯЯ существующие"
                "\nБЕЗ дополнительных вопросов!\nБудем выгружать"
    )
  (progn
    (if (setq folder (_ru-dlg-folder "Выбор папки для записи блоков"
          (ru-user-work-dir) (ru-user-read-last-param
          "block_split_folder" (getvar "DWGPREFIX")) T))
```

```
      (progn
        (ru-user-write-last-param "block_split_folder" folder)
        (if (setq lst_names_descr
             (ru-block-list-all-names-and-decriptions))
          (progn
            (foreach block lst_names_descr
              (setq name (car block) descr (cadr block))
              (if (not descr) (setq descr ""))
              (setq file (strcat (ru-dirs-path-and-slash folder) name))
              (princ (strcat "\nЗаписываю - " file))
              (if (findfile (ru-file-set-ext file ".dwg"))
                        (command "_.WBLOCK" file "_Y" "=")
                        (command "_.WBLOCK" file "=")
             );_ end of if
              (if (/= descr "")
                  (ru-file-write-dirinfo-file-comment file descr)
             );_ end of if
           );_ end of foreach
         );_ end of progn
       );_ end of if
     );_ end of progn
   );_ end of if
  );_ end of progn
);_ end of if
 );_ end of if
(princ)
);_ end of defun
(START)
```

При выполнении этой программы в выбранный каталог записываются файлы блоков с описаниями блоков, сохраненными в файле dirinfo.ini.

Листинг 24.29. Файл ru_block_insert_all.lsp

```
(defun START (/ files_list)
 ;; вставка отобранных файлов
(if (ru-user-action-enabled "DEVELOP" T)
 (if (setq files_list (ru-dlg-file-select-dwg-multi
  "Выбор блоков для вставки" (ru-user-work-dir)
  (ru-user-read-last-param "block_split_folder" (getvar "DWGPREFIX")) T))
    (foreach file files_list
      (ru-error-catch
           (function (lambda ()
                       (princ (strcat "\nВставляю " file))
                       (ru-block-insert-obj file (list 0 0 0) 1 1 1 0)
                  );_ end of lambda
          );_ end of function
           (function (lambda (msg)
                       (princ (strcat "\nОШИБКА вставки: " msg))
                  );_ end of lambda
```

```
          );_ end of function
        );_ end of ru-error-catch
      );_ end of foreach
 );_ end of if
);_ end of if
(princ)
);_ end of defun
(START)
```

В функции вставки файлов мы использовали объектный метод с дополнительной ловушкой ошибок, т. к. при вставке возможно переопределение блоков, встроенных в AutoCAD (например, блоки засечек для размеров).

Расстановка всех блоков

После вставки блоков (а все они вставляются в начало координат) возникает желание расставить их "рядами и колоннами", примерно так, как мы поступили со штриховками. Упорядочивание блоков никогда не повредит, поэтому мы напишем очередную программу (листинг 24.30).

Листинг 24.30. Файл ru_pro_arrange_blocks.lsp

```
(defun START (/ blocksdx dxcurr n name row t0 tx x xcurr xmax ycurr)
 (if (ru-yes (strcat "Программа растаскивает\n ВСЕ БЛОКИ"
                                "\nпо рисунку!\nБудем делать")
  (progn
   (setq blocks (ru-obj-list-blocks)
     t0 (ru-get-point-reguired "Левый верхний угол матрицы блоков" nil)
     dx (ru-get-dist "\nРасстояние между блоками: " 10.0 t0)
     xmax (ru-get-dist "\nРазмер матрицы максимальный: " (* dx 10) t0)
     tx t0 n 0 row 0)
  (foreach name blocks
   (if (/= (substr name 1 1) "*")
     (progn
       (princ (strcat "\nНайден блок - " name))
       (setq dxcurr (* dx n) xcurr  (+ (car t0) dxcurr) ycurr  (cadr tx))
       (if (> dxcurr xmax)
         (setq n 0 row   (1+ row) xcurr (car t0)
              ycurr (- (cadr t0) (* dx row)))
       );_ end of if
       (setq tx (list xcurr ycurr) n  (1+ n))
       (ru-block-insert-obj name tx 1 1 1 0)
     );_ end of progn
   );_ end of if
 );_ end of foreach
);_ end of progn
);_ end of if
```

```
(princ)
);_ end of defun
(START)
```

Запись списка блоков в файл

В связи с тем, что для приведения в соответствие со "стандартом ruCAD" нам потребуется переименовать множество блоков, а затем изменить все старые имена на новые во множестве меню и программ, удобнее всего создать для каждой библиотеки (или любого файла) список блоков и описаний. В дальнейшем эти файлы можно использовать в программе поиска и замены *(см. главу 9)*. Формат строк файла должен быть таким:

```
имя_блока|Примечание к блоку, а если его нет, имя_блока|имя блока
```

В конце строки мы повторяем `имя_блока`, чтобы из него удобнее было сделать новое имя. Изменять имена множества блоков с помощью текстового редактора удобнее, чем в диалоговом окне. Программа записи имен блоков очень проста (листинг 24.31), но значительно облегчает работу.

Листинг 24.31. Файл ru_block_names_file.lsp

```
(defun START (/ file lst_names_descr descr name string_list)
 (if (setq lst_names_descr (ru-block-list-all-names-and-decriptions))
  (progn
   (setq file (ru-file-dwgname-type "blocks"))
   (foreach block lst_names_descr
     (setq name  (car block)
          descr (cadr block)
     );_ end of setq
     (if (not descr) (setq descr ""))
    (setq string_list (cons (strcat name "|" (if (/= descr "") descr
         name) "|" name) string_list))
  );_ end of foreach
   (ru-list-write-to-file file (reverse string_list))
   (ru-msg-info (strcat "Записан файл " file
                "\n с именами и пояснениями к блокам"
                "\nИспользуйте для переименования блоков"
       );_ end of strcat
  );_ end of ru-msg-info
 );_ end of progn
);_ end of if
(princ)
);_ end of defun
(START)
```

После редактирования файла имен и примечаний можно переименовать блоки с помощью программы, приведенной в листинге 24.32.

Листинг 24.32. Файл ru_block_rename_from_file.lsp

```
(defun START (/ err_list err_file)
  (if (setq err_list
            (ru-block-rename-with-descr
              (ru-file-dwgname-type "blocks")
              (ru-file-dwgname-type "replace")
            );_ end of ru-block-rename-with-descr
     );_ end of setq
    (progn
      (setq err_file (ru-file-tmp "~block_rename.err"))
      (if (ru-list-write-to-file err_file err_list)
        (ru-dlg-view-txt-file err_file
         err_file "Ошибки переименования" T)
     );_ end of if
   );_ end of progn
 );_ end of if
  (ru-block-show-all-names-and-decriptions nil)
  (princ)
);_ end of defun
(START)
```

Программа прочитает файл и переименует блоки с присвоением им новых описаний. Если при переименовании обнаруживаются ошибки, то сообщения об ошибках записываются во временный файл, который будет выведен для просмотра. В конце будет показан список имен блоков и описаний. В качестве дополнительной услуги создается файл произведенных замен, который можно использовать в других программах.

Переопределение указанного блока

Часто приходится переопределять блоки с небольшой корректировкой, исключая или добавляя примитивы. Эту операцию удобно выполнять с помощью простой программы (листинг 24.33).

Листинг 24.33. Файл ru_block_redefine.lsp

```
(defun START (/ block_name base_point ent ent_lst selection)
(ru-app-begin)
(if (ru-user-action-enabled "DEVELOP" T)
 (while
  (setq ent (ru-get-entsel-by-type "Выбери блок для переопределения"
                "Это не БЛОК" (list "INSERT") T))
   (setq ent (car ent) ent_lst (entget ent)
         base_point (cdr (assoc 10 ent_lst))
         block_name (cdr (assoc 2 ent_lst))
         base_point (ru-get-point-with-default "Новая точка вставки"
                      base_point base_point))
```

```
    (ru-obj-ent-ss-explode ent)
    (princ (strcat "\nВыберите объекты для блока " block_name ": "))
    (if (setq selection (ru-ss-get))
      (ru-block-obj-make-def block_name base_point selection))
)
)
(ru-app-end)
(princ)
)
(START)
```

Пользоваться этой программой нужно так:

- если из блока надо исключить часть примитивов, достаточно выбрать переопределяемый блок, подтвердить прежнюю точку вставки и выбрать примитивы, не указывая исключаемые;
- если требуется добавить примитивы, удобнее нарисовать возле существующей вставки новые примитивы и выбрать их при формировании набора;
- если требуется изменить геометрию блока, нужно рядом с существующей нерасчлененной вставкой блока нарисовать новые или откорректировать расчлененную вставку, указывать на нерасчлененный блок, а точку вставки и набор примитивов указывать по усмотрению.

Команды BLOCK (БЛОК) и REFEDIT (ССЫЛРЕД) в современных версиях системы AutoCAD очень удобны, с помощью стандартного диалогового окна команды BLOCK можно переопределить блок, но если это нужно делать сотни раз, то выбирать имя переопределяемого блока из списка быстро надоест. Тем более что мы можем и не знать *имя блока*, но знаем и видим, какое *изображение* нам требуется откорректировать.

Просмотр информации об объектах рисунка

Напоследок разработаем несколько простых программ для просмотра свойств объектов — не в окне редактора свойств, а в виде списка с DXF-кодами (рис. 24.11). Не знаем, как обстоят дела у читателей, а нам очень часто приходится набирать в командной строке что-то наподобие `(entget (car (entsel)))`, а потом разбираться с выведенным списком. Давайте напишем программу для вывода этой информации в более читабельном виде. Сначала сделаем это для просмотра свойств любого примитива (листинг 24.34). Данные мы выводим именно в виде точечных пар, не пытаясь выводить осмысленные названия — это прекрасно делает стандартный диалог свойств объектов.

Листинг 24.34. Файл ru_pro_list_entity_data.lsp

```
(defun START (/ ent lst)
 (while (setq ent (ru-get-entsel "Выбери объект"))
  (if (setq lst (ru-conv-list-to-string-list (entget (car ent))))
    (ru-dlg-single-list "Вывод данных примитива" "Данные примитива"
        lst)
```

```
        (print (entget (car ent)))
   )) (princ))
(START)
```

Для просмотра вложенных данных можно использовать функцию, приведенную в листинге 24.35.

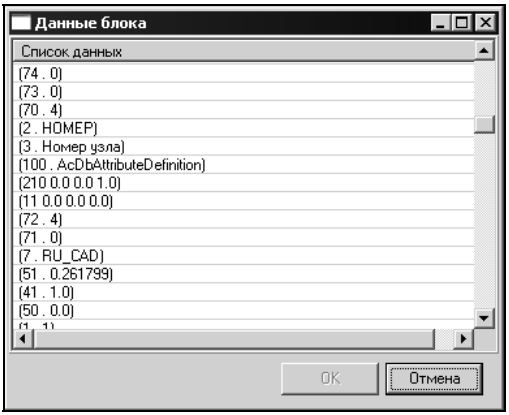


Рис. 24.11. Просмотр списка данных объекта

Листинг 24.35. Файл ru_pro_list_nested_entity_data.lsp

```
(defun START (/ ent lst)
 (while (setq ent (ru-get-nentsel "Выбери объект"))
  (if (setq lst (ru-conv-list-to-string-list (entget (car ent))))
    (ru-dlg-single-list "Вывод данных примитива"
     (if (< (length ent) 3)
       "Данные  примитива" "Данные ВЛОЖЕННОГО примитива") lst)
      (print (entget (car ent)))
   ))(princ))
(START)
```

Эта программа кажется более универсальной, но она не позволяет просмотреть свойства вставки блока, поэтому напишем еще одну программу, "натасканную" на блоки (листинг 24.36).

Листинг 24.36. Файл ru_pro_list_all_block_entity_data.lsp

```
(defun START (/ block_name lst_data s_list _list_block_data)
  (defun _list_block_data (block_name etype / reslist DATA)
    (setq data (tblsearch "block" block_name)
          reslist (cons data reslist)
          data (ru-ent-dxf-code-data -2 data)
          data (entget data '("*")))
    (if etype (setq etype (strcase etype)))
    (while data
      (cond
```

```
        (etype
         (if (= etype (ru-ent-dxf-code-data 0 data))
           (progn
             (setq reslist (cons data reslist))
             (if (setq data (entnext (ru-ent-dxf-code-data -1 data)))
               (setq data (entget data '("*")))
               (setq data nil)
             )
           )
           (setq data
                  (if (setq data
                            (entnext (ru-ent-dxf-code-data -1 data)))
                    (entget data '("*"))
                  )
           )
         )
        )
       )
        (T
         (setq reslist (cons data reslist))
         (if (setq data (entnext (ru-ent-dxf-code-data -1 data)))
           (setq data (entget data '("*")))
           (setq data nil)
         );_ end of if
        )
      )
    )
     (reverse reslist)
 );_ end of defun
;;;-------------------------------------------------------------------------
  (while (setq block_name
           (ru-get-block-name "Выбор блока для просмотра свойств"))
    (if (setq lst_data (_list_block_data block_name nil))
      (progn
        (foreach x lst_data
          (foreach y (ru-conv-list-to-string-list x)
            (setq s_list (cons y s_list))
          )
        )
        (setq s_list (reverse s_list))
        (if s_list (ru-dlg-single-list "Данные блока" "Список данных"
            s_list)
          (print lst_data)
        );_ end of if
      );_ end of if
      (ru-msg-alert "Данные не извлечены")
    )
 )
  (princ)
)
(START)
```

Резюме

В этой главе мы рассмотрели несколько программ, требующихся самим разработчикам. Надеемся, что читатели обратили внимание на то, как часто мы используем библиотечные функции, на разработку которых потратили столько времени. В следующих главах мы продолжим демонстрацию наших технологий, но уже в программах, предназначенных для конечных пользователей.

ГЛАВА 25

Программы общего назначения

В этой главе мы разберем несколько программ общего назначения. Такие программы предназначены для облегчения работы любого специалиста. Хотя наша книга про САПР строительного профиля, но эти программы могут использоваться и машиностроителями. Мы постарались отобрать для демонстрации малоизвестные решения.

Переключение компоновок

При большом количестве *листов* (которые в данной книге будем называть *компоновками*), особенно с длинными именами, переключение между ними стандартным способом неудобно — приходится прокручивать список вкладок. Разработаем программу выбора компоновки из вертикального списка (рис. 25.1).

Вариант такой программы, использующей язык DCL, можно найти в Интернете[1]. Мы пойдем другим путем — используем свое универсальное диалоговое окно выбора из списка, а кроме того, добавим в строку состояния системы AutoCAD специальную кнопку для вызова этого окна (одну такую кнопку-переключатель меню мы уже добавили).

Основная функция установки выбранной компоновки приведена в листинге 25.1.

Листинг 25.1. Функция `ru-dlg-set-layout`

```
(defun ru-dlg-set-layout (/ layout_list)
  (if (setq layout_list (ru-dlg-select-layout))
;;Вариант -(setvar "CTAB" (car layout_list))
    (vla-put-activelayout (ru-obj-get-active-document)
     (car (cadr layout_list)))
 )
 (princ (strcat "\nВключен Layout  " (getvar "CTAB") "\n"))
 (princ)
);_ end of defun
```

[1] **www.afralisp.com/newsletter/code/code76.htm**.

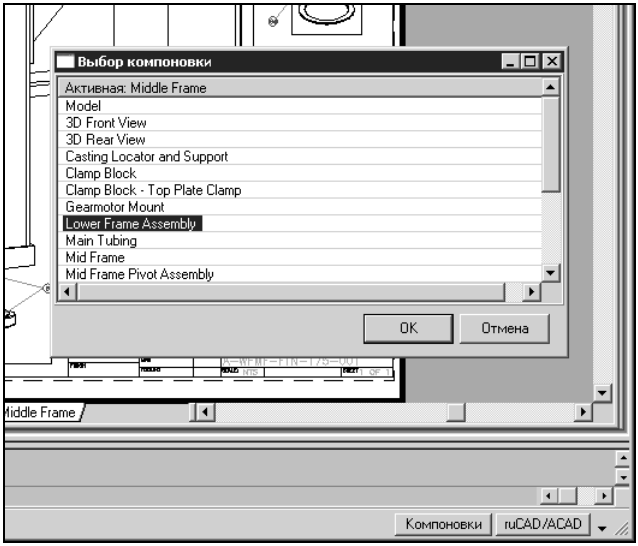


Рис. 25.1. Выбор компоновки из списка

Функция визуального выбора компоновки (листинг 25.2) возвращает список из имени выбранной компоновки и ее VLA-объекта.

Листинг 25.2. Функция `ru-dlg-select-layout`

```
(defun ru-dlg-select-layout (/ layouts_with_name name)
  (setq layouts_with_name (ru-obj-get-layouts-with-name))
  (if (setq name (ru-dlg-single-list "Выбор компоновки"
                   (strcat "Активная: " (getvar "CTAB"))
                   (mapcar 'car layouts_with_name)
                 );_ end of ru-dlg-single-list
    );_ end of setq
    (list name (cdr (assoc name layouts_with_name)))
    nil
 );_ end of if
);_ end of defun
```

Теперь разработаем функцию для добавления кнопки (листинг 25.3).

Листинг 25.3. Функция `ru-acad-sbb-make-layouts-button`

```
(defun ru-acad-sbb-make-layouts-button
       (button_text button_clicked_call_back / srv)
  (setq *ru_sbb_layouts_maked* (vl-bb-ref '*ru_bb_sbb_layouts_maked*))
  (if (not *ru_sbb_layouts_maked*)
    (progn
      (setq srv (vlax-create-object "AcadStatusBarButton.Button"))
      (vlax-put-property srv 'Text button_text)
```

```
      (vlax-put-property srv 'Width 0)
      (vlax-put-property srv 'TooltipText "Выбор компоновки")
      (vlax-put-property srv 'TooltipEnabled :vlax-true)
      (vlax-put-property srv 'LispFunctionOnClicked
        (vl-symbol-name button_clicked_call_back)
     );_ end of vlax-put-property
      (vlax-put-property srv 'Visible :vlax-true)
      (setq *ru_sbb_layouts_maked* t)
      (vl-bb-set '*ru_bb_sbb_layouts_maked* *ru_sbb_layouts_maked*)
   );_ end of progn
 );_ end of if
  srv
);_ end of defun
```

И напоследок добавим код в конец функции `ru-init-start-rucad`:

```
(setq *ru_sbb_layouts_srv* (ru-acad-sbb-make-layouts-button "Компоновки"
'ru-dlg-set-layout))
```

для того чтобы новая кнопка появлялась при запуске системы AutoCAD.

Эти функции работают нормально, но обнаруживается неприятная особенность объекта `AcadStatusBarButton.Button` — после переключения компоновок система AutoCAD делает вид, что зависает и не выдает приглашения на ввод команды. Придется сделать ухудшение кода. Введем промежуточную функцию, реагирующую на нажатие кнопки (листинг 25.4). Мы вынуждены вместо прямого вызова функции `ru-dlg-set-layout` вызывать ее через командную строку.

Листинг 25.4. Функция `ru-acad-sbb-make-layouts-button-on-click`

```
(defun ru-acad-sbb-make-layouts-button-on-click ()
  (vlax-invoke-method(vla-get-activedocument (vlax-get-acad-object))
    'SendCommand "(ru-dlg-set-layout)\n")
(princ)
);_ end of defun
```

Приходится изменить и код функции `ru-init-start-rucad`:

```
(setq *ru_sbb_layouts_srv* (ru-acad-sbb-make-layouts-button "Компоновки"
 'ru-acad-sbb-make-layouts-button-on-click))
```

Вот теперь все нормально, только "мусор" в командной строке остается.

Быстрое стирание

Иногда приходится стирать много мелких примитивов, да еще и "неправильно" расположенных. Хотелось бы быстро их стирать, просто проводя мышью по рисунку, как это делается в графических редакторах растровых файлов. В качестве "ластика" можно было бы использовать прицел выбора объектов.

Для этого нужно непрерывно отслеживать перемещение указателя мыши и нажатия клавиш. Решается такая задача с помощью функции `grread`. Все варианты использо-

вания этой функции мы разбирать не будем. Для наших целей пригоден следующий формат вызова — `(grread T 4 2)`, соответствующий условиям:

- отслеживать положение курсора (значение первого аргумента — `T`);
- отображать курсор (значение второго аргумента — 4) в виде прицела выбора объектов (значение третьего аргумента — 2).

Функция `grread` возвращает список, первым элементом которого будет код, указывающий тип ввода, а вторым элементом — либо список с координатами точки, либо целая величина. Нас интересуют только типы ввода, соответствующие величине первого элемента списка: 5 (перемещение указателя), 3 (указание точки) и 2 (ввод с клавиатуры). При коде 5 или 3 список будет наподобие `(5 (896.7 1680.48 0.0))`, а при коде 2 — `(2 43)`. Целая величина при коде 2 соответствует коду нажатой клавиши.

Опасность применения такой функции при выборе из меню в том, что стирание будет происходить сразу же, в том числе пока курсор перемещается к задуманному месту. Следовательно, необходимо явно начинать и останавливать стирание. Самым естественным событием для начала и приостановки стирания был бы щелчок левой кнопкой мыши (как бы указание первого стираемого примитива), а полным завершением — нажатие клавиши <Enter> (на самом деле любой клавиши или правой кнопки мыши). Текст программы, выполняющей быстрое стирание, приведен в листинге 25.5.

Листинг 25.5. Файл ru-rubber.lsp

```
(defun START (/ key mode point selection *error*)
;;; Локальный обработчик ошибок
  (defun *error* (msg)
    (ru-obj-undo-end)
    (command "_.U")
    (princ "\nСтирание отменено\n")
    (princ)
 );_ end of defun

  (ru-obj-undo-begin)
  (setq key 3 mode T)
  (while (or (= key 3) (= key 5))
    (cond ((= key 3)
           (princ
;|-------------------------------------------------------------------------
Выход показываем в виде результата действия по умолчанию, т. е. нажатия клавиши
Enter. Фактически можно нажать любую клавишу, но не будем смущать "теток"
двусмысленным предложением "Any key"
-------------------------------------------------------------------------|;
             (if (setq mode (not mode))
               "\nРежим ЛАСТИК (клик = приостановить стирание) <Выход>"
               "\nУкажи точку для запуска режима ЛАСТИК <Выход>"
             )
          )
         )
          ((not mode))
```

```
          (T
           (if (setq selection (ssget (cadr point)))
             (entdel (ssname selection 0))
           );_ end of if
          )
   );_ end of cond
    (setq key (car (setq point (grread T 4 2))))
 );_ end of while
  (princ "\nРабота команды завершена.")
  (ru-obj-undo-end)
  (princ)
);_ end of defun
(START)
```

Замечание

Мы делали и "продвинутый" вариант этой программы — с возможностью динамического изменения размера курсора нажатием клавиш <+> или <–>. Но пользоваться такой программой оказалось так же неудобно, как водить автомобиль, у которого имеются несколько дополнительных педалей.

Программа, по нашему мнению, достойна включению в панель инструментов. Напоминаем, как это делается в системе ruCAD:

```
[_Button("Ластик - быстрое стирание под курсором", RU_RUBBER_16,
RU_RUBBER_32)]^C^C^P(defun C:RU()(ru-app-load "ru_rubber"));RU
```

Обратите внимание на то, что мы переопределили обработчик ошибок, а в нем, в отличие от нашей обычной практики, предусмотрели отмену изменений. Это "опасная" программа и "тетка", испугавшись быстрого пропадания объектов, инстинктивно нажимает клавишу <Esc>. В этом случае стирание будет отменено.

Текстовая "лупа"

Очень часто приходится выполнять зумирование только для того, чтобы прочитать мелкую надпись.

Замечание

В системе AutoCAD вообще не хватает "лупы", позволяющей просматривать увеличенный фрагмент. А ведь такая возможность была в версии AutoCAD R13c4 для Windows. Там окно общего вида, вызываемое по команде DSVIEWER, имело инструментальную функцию **Фрагмент**. При этом действительно в окне общего вида показывался фрагмент изображения возле курсора. То есть окно общего вида работало наоборот. Да еще масштабом увеличения можно было управлять. И было это очень удобно!

Очень просто сделать лупу для увеличения фрагмента экрана, но только в растровом виде. Можно использовать и штатный Magnifier из комплекта Windows, но проку от таких "луп" мало.

Просматривать любые надписи (тексты, размеры, значения атрибутов) можно очень простой программой (листинг 25.6).

Листинг 25.6. Файл ru_view_text_magnitify.lsp

```
(defun START (/ txt)
  (while (setq txt (ru-get-txt-from-dwg "Выбери надпись для просмотра"))
    (princ (strcat "\nТекст указанной надписи: '" txt "'\n"))) (princ)
)
(START)
```

Такую программу лучше "привязать" к кнопке на панели инструментов, иначе проще будет зумировать изображение, чем разыскивать инструмент в меню.

Быстрое рисование "такого же" объекта

Очень часто пользователю нужно нарисовать такой же объект, какой он видит — на таком же слое, с таким же цветом и прочими характеристиками. Простое копирование не всегда пригодно — это может быть и линия, у которой должен быть другой контур. Да и простое копирование при быстрой работе неудобно — нужно выбрать объект, указать базовую точку, не забыв про возможную опцию **Multiple** (Несколько), и только потом указывать точки вставки. Да и "хвост" резиновой нити будет раздражать. Разработаем еще одну простую, но эффективную программу (листинг 25.7). Для повышения динамики работы рисование "таких же" объектов предусматривается с использованием циклов.

Листинг 25.7. Файл ru_draw_by_object.lsp

```
(defun START (/elist ename etype list_sysvars_codes point sysvars_list
  val _circle _text _hatch _block _command _copy-paste)
;|-------------------------------------------------------------------------
Локальная функция копирования-вставки примитива. Мы не любим использовать буфер
обмена, но в данной ситуации считаем это возможным
-------------------------------------------------------------------------|;
  (defun _copy-paste (ename point)
  (command "_.COPYBASE" point ename "") (_command "_.PASTECLIP"))
;|-------------------------------------------------------------------------
Локальная функция. Имитатор любой стандартной команды
-------------------------------------------------------------------------|;
  (defun _command (command_name / do_it old_echo)
   (setq old_echo (getvar "CMDECHO") do_it t)
    (setvar "CMDECHO" 1)
    (while do_it
      (vl-cmdf command_name)
      (while (> (getvar "CMDACTIVE") 0) (vl-cmdf pause))
;|-------------------------------------------------------------------------
Лучше вставить дополнительный запрос в командной строке, чем заставлять вновь
запускать программу
-------------------------------------------------------------------------|;
      (setq do_it (ru-dlg-yes-cml (strcat "Повторить?")))
   )
    (setvar "CMDECHO" old_echo)
 )
```

```
;|-------------------------------------------------------------------------
Локальная функция. Вставка "такого же" блока
-------------------------------------------------------------------------|;
  (defun _block (elist ename point / name)
    (if (= (substr (setq name (cdr (assoc 2 elist))) 1 1) "*")
;; Анонимные блоки и штриховки
      (_copy-paste ename point)
;; Обычные именованные блоки
      (ru-block-multi-insert-scaled-rotated-or-angleask
        name (cdr (assoc 41 elist)) (cdr (assoc 42 elist))
        (ru-conv-rad-to-deg (cdr (assoc 50 elist)))
     )
   );_ end of if
 );_ end of defun
;|-------------------------------------------------------------------------
Локальная функция. Рисование текстов
-------------------------------------------------------------------------|;
(defun _text (elist ename / string height justification point ang point_1 point_2
point_a)
  (setq height (cdr (assoc 40 elist)) ang (cdr (assoc 50 elist))
      string (cdr (assoc 1 elist)) justification (cdr (assoc 72 elist)))
  (cond
   ((or (= justification acAlignmentAligned)
        (= justification acAlignmentFit))
    (setq point_1 (cdr (assoc 10 elist)) point_2 (cdr (assoc 11 elist)))
  )
 )
;|-----------------------------------------------------------------
"Точно такой же" текст редко нужен, даем возможность изменить текстовую строку
-----------------------------------------------------------------|;
  (while (setq string (ru-get-string "Новый текст" string))
   (while
    (setq point
      (ru-get-point-or-exit (strcat "Точка текста '" string "' ") nil))
    (ru-text-add string
     (cond
      ((or (= justification acAlignmentAligned)
           (= justification acAlignmentFit))
          ;; точка 2 берется как от точки1 до точки2
          (setq point_1 (trans point_1 ename 0)
                point_2 (trans point_2 ename 0)
                ang     (angle point_1 point_2)
                point_a (polar point ang (distance point_1 point_2))
         )
          (list  point point_a)
        )
        (T point)
       )
     height ang justification)))
)
```

```
;|-------------------------------------------------------------------------
Локальная функция. Рисование кругов
-------------------------------------------------------------------------|;
  (defun _circle (elist / r)
  (setq r (cdr (assoc 40 elist)))
  (while
   (vl-cmdf "_.CIRCLE" (ru-get-point-or-exit "Центр круга" nil) r))
 )
;|-------------------------------------------------------------------------
Основная функция
-------------------------------------------------------------------------|;
 (ru-app-begin)
 (if (setq ename (ru-get-entsel
                   "Выбери объект-образец для создания таких же"))
  (progn
;|-------------------------------------------------------------------------
Запоминаем точку указания - пригодится в качестве базовой для копирования и вставки
-------------------------------------------------------------------------|;
      (setq point              (cadr ename)
            ename              (car ename)
;|-------------------------------------------------------------------------
Список свойств выбранного примитива
-------------------------------------------------------------------------|;
            elist              (entget ename)
            etype              (strcase (cdr (assoc 0 elist)))
;|-------------------------------------------------------------------------
Начинаем формировать список системных переменных, которые нужно запомнить и
восстановить. Сначала составляем список переменных, не зависящих от типа примитива
-------------------------------------------------------------------------|;
            list_sysvars_codes
            (list (cons 6 "CELTYPE") (cons 7 "TEXTSTYLE")
                  (cons 8 "CLAYER") (cons 39 "THICKNESS")
                  (cons 62 "CECOLOR") (cons 370 "CELWEIGHT")
            );_ end of list
     );_ end of setq
;|-------------------------------------------------------------------------
Добавляем коды и переменные, зависящие от типа примитива
-------------------------------------------------------------------------|;
   (cond
     ((= etype "HATCH")
      (setq list_sysvars_codes
        (append
          (list (cons 2 "HPNAME") (cons 41 "HPSCALE") (cons 52 "HPANG"))
          list_sysvars_codes);_ end of append
        );_ end of setq
    )
    ((or (= etype "POLYLINE") (= etype "LWPOLYLINE"))
      (setq list_sysvars_codes
                (append (list (cons 43 "PLINEWID")) list_sysvars_codes)
      );_ end of setq
   )
```

```
    ((= etype "TEXT")
      (setq list_sysvars_codes
            (append (list (cons 40 "TEXTSIZE")) list_sysvars_codes)
     );_ end of setq
  )
  );_ end of cond
;|-------------------------------------------------------------------------
Список общих системных переменных
-------------------------------------------------------------------------|;
  (setq sysvars_list
   (list (list "CMDECHO" 0) (list "BLIPMODE" 0) (list "OSMODE")
   (list "PICKBOX") (list "HIGHLIGHT") (list "ORTHOMODE"))
    );_ end of setq
;|-------------------------------------------------------------------------
Объединяем все запоминаемые переменные в один список
-------------------------------------------------------------------------|;
  (mapcar
  (function
    (lambda (x) (if (setq val (cdr (assoc (car x) elist)))
     (setq sysvars_list (cons (list (cdr x) val) sysvars_list))
     (setq sysvars_list (cons (list (cdr x)) sysvars_list))
    )))
       list_sysvars_codes
 );_ end of mapcar
;|-------------------------------------------------------------------------
Запоминаем список переменных и их значений, как это делается в обработчике ошибок.
Теперь, при ошибке, они будут восстановлены
-------------------------------------------------------------------------|;
  (ru-error-save-sysvars sysvars_list)
;|-------------------------------------------------------------------------
Для каждого типа примитива-образца запускаем соответствующую функцию
-------------------------------------------------------------------------|;
 (cond
  ((= etype "CIRCLE") (_circle elist))
  ((= etype "HATCH")
 ;; Вариант стандартной команды (_command "_.BHATCH")
  (ru-hatch-draw-pattern (cdr (assoc 2 elist)) (cdr (assoc 41 elist))
          (cdr (assoc 52 elist))))
  ((= etype "INSERT") (_block elist ename point))
  ((= etype "LINE") (_command "_.LINE"))
  ((or (= etype "POLYLINE") (= etype "LWPOLYLINE")) (_command "_.PLINE"))
  ((= etype "TEXT") (_text elist ename))
;|-------------------------------------------------------------------------
Для любого другого типа
-------------------------------------------------------------------------|;
  (T (_copy-paste ename point))
);_ end of cond
;|-------------------------------------------------------------------------
Восстанавливаем значения переменных
-------------------------------------------------------------------------|;
      (ru-error-restore-sysvars)
  );_ end of progn
 );_ end of if
```

```
  (ru-app-end)
  (princ)
);_ end of defun
(START)
```

Опыт показывает, что как только пользователи понимают, что делает эта функция, кнопка **Такой же** на панели инструментов становится одной из самых любимых.

Программы для быстрого штрихования

При разработке некоторых разделов проекта штриховать приходится много, и стандартное диалоговое окно, открываемое командой BHATCH (КШТРИХ), в котором можно сделать какую угодно штриховку, становится тормозом в работе. При штриховании возникают две основных проблемы — выбор нужного образца и установка правильного масштаба.

Традиционно *выбор образца* производился из слайдового меню, сейчас используется диалоговая команда BHATCH (КШТРИХ). Стандартный выбор образца из списка с последующим просмотром в микроскопическом поле является издевательством над пользователями. Кроме того, пользователю не важны маловразумительные названия, ему нужно выполнять штрихование конкретного изображения (металл или кирпич в разрезе), и можно использовать один образец, но с разными масштабами и углами поворота для различных изделий.

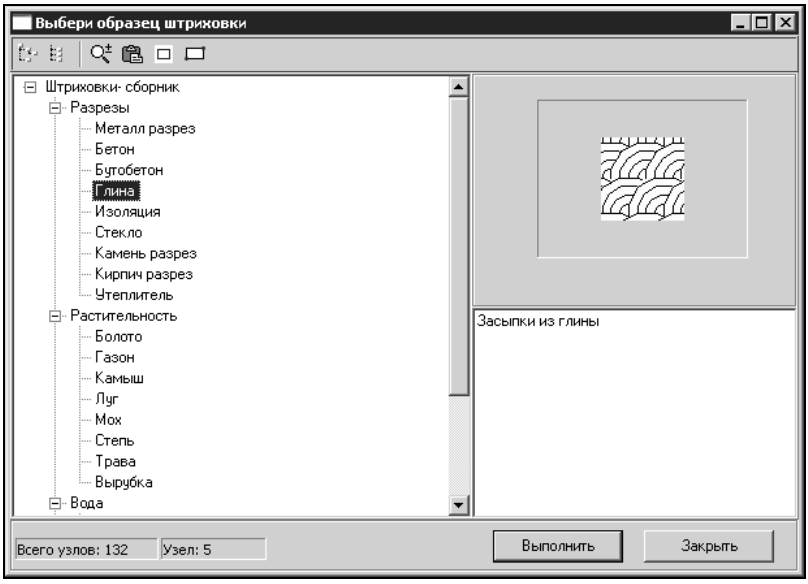


Рис. 25.2. Выбор образца штриховки

Разумеется, мы предусмотрели выбор образца из иллюстрированного XML-меню (рис. 25.2). На рисунке показано диалоговое окно, выводимое при выборе опции

XML во время работы функции `ru-hatch-draw-pattern` *(см. главу 22)*. Когда пользователю нужно нарисовать заведомо штрихуемый объект (например, контуры зон различного назначения на генплане), то он должен иметь возможность выбирать не образец штриховки, а команду, внутри которой создается штриховка с заданными параметрами.

В результате программирование рисования изображений со штриховками может свестись к разработке нескольких простеньких вариантов использования типовой функции штрихования и к формированию XML-меню.

Установка масштаба штриховки также должна быть упрощена. Для конкретных изображений заранее определенный масштаб указывается в аргументах функций, а для "свободной штриховки" оптимальный масштаб у нас определен заранее и сохранен для каждого образца в файле hatch.ini (о том, как установить оптимальный масштаб штриховок, мы писали в *главе 24*).

Универсальная программа свободной штриховки приведена в листинге 25.8.

Листинг 25.8. Файл ru_hatch_draw.lsp

```
(defun START (/ hatch_name)
  (ru-app-begin)
  (if (setq hatch_name (ru-get-hatch-name "Выбор образца штриховки"))
    (ru-hatch-draw-pattern hatch_name (ru-get-hatch-scale hatch_name) 0)
 );_ end of if
  (ru-app-end)
  (princ)
)
(START)
```

Пример специализированной программы рисования контуров зданий приведен в листинге 25.9.

Листинг 25.9. Файл ru_gp_house_hatch.lsp

```
(defun START (pattern scale ang / ss)
  (ru-app-begin)
  (if (setq ss (ru-pline-make-multi-contour "штриховки здания"))
    (ru-hatch-draw pattern scale ang ss)
 );_ end of if
  (ru-app-end)
  (princ)
);_ end of defun
```

Пример вызова этой программы из XML-меню показан в листинге 25.10.

Листинг 25.10. Файл hatch_zd.ruxm

```
<?xml version='1.0' encoding='windows-1251'?>
<root name='Назначение зданий' macro=''>
 <item name='Здание реконструируемое' image='GP\ZD_REC.GIF'
  comment='Контур здания со штриховкой' macro=
```

```
'(progn(if(ru-app-load "ru_gp_house_hatch")(START "ansi31" 15 0)))'/>
  <item name='Здание 1-я очередь до 5 этажей' image='GP\ZD_1_1.GIF'
  comment='Контур здания со штриховкой' macro=
'(progn(if(ru-app-load "ru_gp_house_hatch")(START "ansi31" 15 45)))'/>
  <item name='Здание 1-я очередь свыше 5 этажей' image='GP\ZD_1_5.GIF'
  comment='Контур здания со штриховкой' macro=
'(progn(if(ru-app-load "ru_gp_house_hatch")(START "ansi31" 30 45)))'/>
  <item name='Здание общественное' image='GP\ZD_OB.GIF'
  comment='Контур здания со штриховкой' macro=
'(progn(if(ru-app-load "ru_gp_house_hatch")(START "ansi37" 20 0)))'/>
  <item name='Здание опорное' image='GP\ZD_OP.GIF'
  comment='Контур здания со штриховкой' macro=
'(progn(if(ru-app-load "ru_gp_house_hatch")(START "ansi32" 10 0)))'/>
</root>
```

Иногда требуется изменить "густоту" существующей штриховки. Разумеется, можно сделать это посредством диалогового окна свойств, а можно и проще, специальной программой (листинг 25.11). Чуть более сложную функцию, предназначенную для подбора оптимального масштаба с записью в файл, мы рассмотрели в *главе 24*.

Листинг 25.11. Файл ru_ed_hatch_scale.lsp

```
(defun START (m / ename old_scale)
;;; Изменение густоты штриховки в M раз
  (ru-app-begin)
  (setvar "HIGHLIGHT" 1)
  (while (setq ent (ru-get-entsel-by-type "Выбери штриховку"
   "Это не ШТРИХОВКА" (list "HATCH") t))
    (setq ename (car ent))
    (if (setq old_scale (cdr (assoc 41 (entget ename))))
        (command "_.-HATCHEDIT" ename "_P" "" (* old_scale m) "")
     ;; это SOLID
        (princ "\nУ этой штриховки масштаб изменять нельзя!\n")
   );_ end of if
 );_ end of while
  (ru-app-end)
  (princ)
);_ end of defun
```

Замечание

Изменение масштаба штриховки с помощью функций `entmod` и `entupd` не получается — свойства меняются, а изображение остается прежним, поэтому применена команда HATCHEDIT (РЕДШТРИХ).

Псевдоштриховка с помощью блока

При всем многообразии образцов штриховок нужную можно не найти. В этом случае может помочь "штриховка" с помощью блока (листинг 25.12).

Листинг 25.12. Файл ru_draw_hatch-block.lsp

```
(defun START (Header mb bn scale dx dy chez ud)
  (ru-draw-block-hatch Header mb bn scale dx dy chez ud)
  (princ)
)
```

Программа является "оберткой" к основной функции штрихования блоком (листинг 25.13).

Листинг 25.13. Функция ru-draw-block-hatch

```
(defun ru-draw-block-hatch (Header block_lib block_name scale dx dy is_chess
ang_deg / contour_ent LSTBOX LSTVER LX LY NX NXT NY NYT SM T0
TMAX TT X X0 XMAX Y Y0 YMAX)
;|-------------------------------------------------------------------------
 Штриховка с помощью блока. Аргументы:
 Header  - заголовок
 block_lib - библиотечный блок
 block_name - блок
 scale - масштаб блока мм/бум
 dx  - интервал по X мм/бум
 dy  - интервал по У мм/бум
 is_chess  - признак шахматности
 ang_deg - поворот блока
Примеры:
(ru-draw-block-hatch "Штриховка блоком" "ru-lib-topo"  "ДЕРЕВО_ШИРОКОЛИСТВ_ТОПО"  1
8 8 T 0)
(ru-draw-block-hatch "Штриховка блоком" "ru-lib-common" "ru_circle_and_cross_unit"
1  3  3 NIL 45)
-------------------------------------------------------------------------|;
 (ru-app-begin)
 (if (ru-block-lib-insert block_lib block_name)
  (progn
    (setq dx    (ru-conv-millimeter-in-paper-to-unit dx)
          dy    (ru-conv-millimeter-in-paper-to-unit dy)
         scale (ru-conv-millimeter-in-paper-to-unit scale))
    (if is_chess (setq sm (/ dy 2.0)) (setq sm 0))
    (if (setq lstVer (ru-pline-make-any-contour Header t NIL))
      (progn (princ "\nЖдите, рисую...\n")
        (setq contour_ent (entlast)
              lstBox (ru-geom-bound-box (car lstVer))
              t0 (cadr lstbox) x0 (car t0) y0 (cadr t0)
              tmax (car lstbox) xmax (car tmax) ymax   (cadr tmax)
              x x0 y y0 lx (- xmax x0) ly (- ymax y0)
              nx (fix (/ lx dx)) ny (fix (/ ly dy)) nxt 0 nyt 0)
        (repeat (1+ nx)
          (setq nxt (1+ nxt))
            (repeat (1+ ny)
              (setq nyt (1+ nyt) tt  (list X Y))
```

```
                (if (ru-match-is-even nyt) (setq tt (polar tt 0 sm)))
                (if (ru-geom-is-point-in-contour tt (car lstVer))
                  (ru-block-insert-obj block_name tt scale scale scale
                     (ru-conv-deg-to-rad ang_deg)))
                (setq Y (+ Y dy))
              )
              (setq y y0 nyt 0 X (+ x dx))
            )
        (if (ru-dlg-yes-cml "Стереть контур") (entdel contour_ent))
        );_ end of progn
       );_ end of if
      );_ end of progn
    );_ end of if
     (ru-app-end)
     (princ)
    );_ end of defun
```

В результате можно получить изображение, подобное показанному на рис. 25.3.

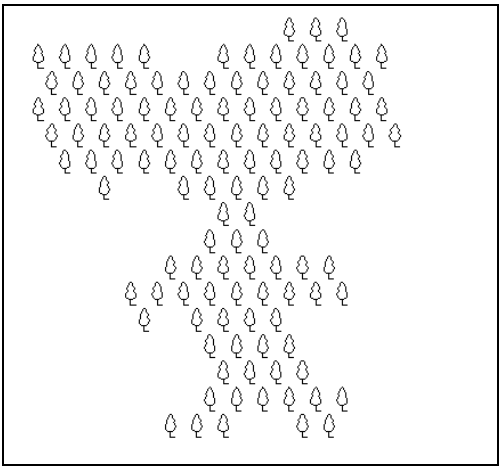


Рис. 25.3. Псевдоштриховка с помощью блока

Работа со слоями

Выполнять различные действия со слоями приходится очень часто. Стандартные средства AutoCAD достаточно удобны, но только пока слоев в рисунке немного. Как только количество слоев начинает превышать десяток, работа с ними становится мучительной. Особенно это касается простых операций — включение, замораживание, блокирование. У нас же еще имеется классификатор слоев, для работы с которым мы уже рассмотрели несколько функций *(см. главу 18)* — запуск классификатора, выбор слоя, получение дополнительных данных для классифицированного слоя, вывод списка слоев с комментариями, выбор слоя указанием. Теперь разработаем несколько программ для повышения динамичности работы.

Установка слоя по образцу

Это была одна из первых программ, которую разрабатывали, наверное, все программисты, пока фирма Autodesk не включила ее в состав "полустандартных". Так как наличие команды AI_MOLC не гарантируется, мы включим в наше меню собственный аналог (листинг 25.14), "привязав" его к привычной всем кнопке.

Листинг 25.14. Файл ru_layer_set_by_object.lsp

```
(defun START (/ layer)
 (if (setq layer (ru-get-layer-from-dwg "Укажи объект в желаемом слое"))
  (ru-layer-current layer))(princ))
(START)
```

Отключение слоя

Так же проста, но эффективна программа отключения указанного слоя (листинг 25.15) — зачем искать нужный слой для отключения в длинном списке, если можно просто указать на объект. Аналогичные программы написаны и для блокировки и замораживания слоев.

Листинг 25.15. Файл ru_layer_off_by_object.lsp

```
(defun START (/ layer)
 (while
   (setq layer (ru-get-layer-from-dwg "Укажи объект в отключаемом слое"))
     (ru-layer-toggle-on-off layer)
)(princ))
(START)
```

Включение всех слоев

Любой, кто хоть раз мучился с "включением лампочек" на нескольких десятках слоев, поймет, как облегчает работу программа, приведенная в листинге 25.16.

Листинг 25.16. Файл ru_layer_show_all.lsp

```
(defun START ()
    (vlax-for each (ru-obj-get-layers)
        (vla-put-layeron each :vlax-true))
(princ "\nВсе слои включены")(princ))
(START)
```

Оставить видимыми указанные слои

Следующая программа (листинг 25.17) имеет противоположное назначение — оставляет видимыми только указанные слои.

Листинг 25.17. Файл ru_layer_stay_selected.lsp

```
(defun START (/ layer layer_list)
 (while (setq layer (ru-get-layer-from-dwg
  "Укажи объект в ОСТАВЛЯЕМОМ слое. Последний указанный станет текущим"))
  (princ (strcat "\nОставляем слой " layer))
  (if (not (member layer layer_list))
     (setq layer_list (cons layer layer_list)))
)
;; Сразу устанавливаем текущим последний слой - в списке он первый
 (ru-layer-current (car layer_list))
 (vlax-for each (ru-obj-get-layers)
   (if (not (member (ru-obj-get-obj-name each) layer_list))
        (vla-put-layeron each :vlax-false)))(princ)
)
(START)
```

Перелистывание слоев

Следующая программа (листинг 25.18) позволяет просматривать слои по одному, отключая все остальные. При таком просмотре пользователь сразу же увидит объекты, находящиеся не на своих слоях. Во время просмотра можно оставлять любой слой видимым, наглядно формируя требуемый набор.

Листинг 25.18. Файл ru_layer_show_by_one.lsp

```
(defun START (/ layer_list layer_name layer_num opt done last_layer)
  (setq layer_list (ru-obj-list-layers))
;; Отключаем все слои
  (vlax-for each (ru-obj-get-layers) (vla-put-layeron each :vlax-false))
(setq layer_name (getvar "CLAYER")
 layer_num  (vl-position layer_name layer_list) last_layer layer_name)
 (while (not done)
;;; включаем слой
  (ru-layer-on layer_name)
  (setq opt (ru-get-kword (strcat "СЛОЙ: " layer_name)
     "Prev Next Stay Exit" "Next"))
  (cond
    ((= opt "Prev")
   ;; Отключаем
     (ru-layer-off layer_name)
    ;;Берем предыдущий номер слоя
     (setq layer_num (1- layer_num))
    ;; Если достигли начала, переходим в конец
     (if (< layer_num 0) (setq layer_num (1- (length layer_list))))
   )
    ((= opt "Next")
     (ru-layer-off layer_name)
     (setq layer_num (1+ layer_num))
     (if (>= layer_num (length layer_list)) (setq layer_num 0))
   )
```

```
   ((= opt "Stay")
   ;; Оставляем слой включенным и удаляем из списка
    (setq layer_list (vl-remove layer_name layer_list)
       layer_num (1+ layer_num) last_layer layer_name)
   (if (>= layer_num (length layer_list)) (setq layer_num 0))
  )
   (T (ru-layer-off layer_name) (setq done T))
 );_ end of cond
  (setq layer_name (nth layer_num layer_list))
  (if (< (length layer_list) 1) (setq done T))
 )
 ;; На выходе делаем текущим последний оставленный слой
  (ru-layer-current last_layer)
  (princ)
)
(START)
```

Стирание слоя

Программа стирания объектов на слое также входит в джентльменский набор начинающего программиста. Действительно, это удобный инструмент, ускоряющий работу. Попробуем развить традиционное решение. Начнем с функции стирания объектов в заданном слое (листинг 25.19). Функция стирания вернет количество удаленных объектов, которое, кстати, может быть нулевым, т. к. если слой будет заблокирован, то объектная функция `ru-obj-ent-ss-erase` отбросит объекты на заблокированных слоях и сообщит об ошибке автоматизации. В данном случае мы не занимаемся проверкой блокировки слоя, поскольку можем обеспечивать проверки на верхнем уровне (листинг 25.20).

Листинг 25.19. Функция `ru-layer-delete`

```
(defun ru-layer-delete (name / ss result)
 (if (setq ss (ssget "_X" (list (cons 8 name))))
  (progn (setq result (sslength ss))
   (ru-obj-ent-ss-erase ss)
   (if (setq ss (ssget "_X" (list (cons 8 name))))
    (setq result (- result (sslength ss))))))
  result
)
```

Конечную программу стирания объектов в слое (листинг 25.20) разработаем с проверкой и подготовкой условий для стирания.

Листинг 25.20. Файл ru_layer_erase.lsp

```
(defun START (/ name n do)
 (ru-app-begin)
 (while (setq name (ru-get-layer-name "Слой для СТИРАНИЯ" nil))
  (setq do T)
```

```
  (if (ru-layer-is-lock name)
    (if (ru-yes (strcat "\nСлой \n'" name "'\n заблокирован "
                  "\nРазблокировать и стереть"))
          (ru-layer-unlock name)
          (setq do nil)))
  (if do (if (setq n (ru-layer-delete name))
      (princ (strcat
        "\nСтерто " (ru-string-number-end n "объект" "а" "ов" "ов")
                    " из слоя '" name "'"))))
 (ru-app-end)
 (princ)
)
(START)
```

Как видите, даже в такой простой программе мы умудрились использовать восемь библиотечных функций, повышающих надежность и удобство работы. Например, выбрать слой можно указанием на примитив и из комментированного списка.

А теперь напишем программу удаления слоя, т. е. стирания всех объектов и ликвидацией определения слоя в рисунке. Сначала разработаем функцию удаления (листинг 25.21).

Листинг 25.21. Функция `ru-layer-purge`

```
(defun ru-layer-purge (name / obj ss)
;; Если слой заблокирован - разблокируем его
  (if (ru-layer-is-lock name) (ru-layer-unlock name))
;; Если на слое есть примитивы — сотрем
  (if (setq ss (ssget "_X" (list (cons 8 name)))) (ru-layer-delete name))
;; Если слой текущий — сделаем активным слой 0
  (if (= (getvar "CLAYER") name) (setvar "CLAYER" "0"))
 (if (setq obj (ru-obj-layer-by-name name))
  (ru-error-catch
   (function (lambda ()(vla-delete obj) t))
   (function (lambda (x)
     (princ (strcat "\nОШИБКА RU-LAYER-DELETE: " x)) nil))
) nil
)
)
```

Удаление слоя мы выполнили объектным методом, предусмотрев традиционную ловушку ошибок с сообщением в случае ошибки. Основная программа удаления слоя у нас не будет зациклена и будет выдавать дополнительный запрос (листинг 25.22).

Листинг 25.22. Файл ru_layer_purge.lsp

```
(defun START (/ name)
 (ru-app-begin)
 (if (setq name (ru-get-layer-name "Слой для полного УДАЛЕНИЯ" nil))
```

```
 (if (not (ru-no (strcat "\nПрограмма полностью УДАЛЯЕТ слой \n" name
"\nи все объекты на нем с удалением из таблицы слоев\nБудем делать")))
   (ru-layer-purge name)))
 (ru-app-end)
(princ)
)
(START)
```

Общие средства рисования

Многие изображения используются при разработке любых частей проекта — различные линии, контуры, тексты, отметки, уклоны, обозначения видов и разрезов, выноски позиций и диаметров, таблицы и т. п. Все эти изображения легко программируются вообще и очень легко — с использованием разработанных нами библиотек базовых функций. Важно обеспечить удобный доступ пользователя к средствам рисования. Самые "ходовые" программы можно привязать к кнопкам панелей инструментов, но экранное место ограничено, а представления о полезности того или иного инструмента у всех разные. Можно заготовить описания панелей на все программы, но как же нам "прокормить" столько "батраков", сколько потребуется хотя бы для рисования 10 000—15 000 миниатюр для кнопок? Как всегда, мы видим выход в использовании иллюстрированных XML-меню.

Особенности XML-меню позволяют вставлять одни и те же макросы в разные меню. Например, рисование высотных отметок может быть и в меню общих средств рисования, и в меню фасадов, разрезов и схем трубопроводов.

Далее мы рассмотрим несколько примеров часто используемых изображений.

Рисование специальных линий

В *главе 22* мы разбирали различные способы рисования трасс. В пользовательских программах рассмотренные функции заключаются в небольшую оболочку. Например, программа `ru_draw_trass_ltype` (листинг 25.23) позволяет рисовать объект заданным типом линии.

Листинг 25.23. Файл ru_draw_trass_ltype.lsp

```
(defun START (ltype lineweight msg can_chanche_lw arc_enabled / old_ltype)
 (ru-app-begin) (setq old_ltype (getvar "CELTYPE"))
 (ru-ltype-set-current ltype)
  (ru-trass-draw-lw-from-pt1-arc
        msg arc_enabled can_chanche_lw lineweight 0)
  (ru-ltype-set-current old_ltype) (ru-app-end) (princ)
)
```

Может возникнуть вопрос — зачем вообще нужны такие программы? Можно просто установить нужный тип линии и рисовать стандартными средствами. Конечно, можно, но:

- нужно знать, какой именно тип линии устанавливать для рисования конкретного изображения (одних пунктирных линий с разным соотношением длины штриха и пропуска имеется очень много);

- требуемый тип линии должен быть загружен;
- после рисования объекта необходимо восстановить ранее действовавший тип линии.

Все эти задачки и автоматизирует приведенная программа. Конкретное ее применение может быть описано в XML-меню (листинг 25.24).

Листинг 25.24. Фрагмент XML-меню контуров зданий

```
<item name='Строящееся' image='TOPO\STRO\STR_STR.GIF'
 comment='Контур строящегося здания тонкой пунктирной линией'
 macro='(progn(if(ru-app-load "ru_draw_trass_ltype")(start "HIDDEN" 0 "контура
строения" NIL NIL)))'/>
<item name='Отмостка' image='TOPO\STRO\OTMOST.GIF'
comment='Контур отмостки здания тонкой пунктирной линией'
macro='(progn(if(ru-app-load "ru_draw_trass_ltype")(start "HIDDEN2" 0
"контура отмостки" NIL NIL)))'/>
```

Два объекта рисуются пунктирной линией, но с разной длиной штриха. Пользователю вообще не нужно знать никаких технических деталей — он выбирает не тип линии, а изображаемый объект.

Еще один пример (листинг 25.25) — рисование трасс текстовым типом линии. Эта программа вроде бы вообще ничего своего не делает, только вызывает резидентную функцию. Однако кое-что делает функция-загрузчик `ru-app-load` *(см. главу 22)* и базовой функции передается часть постоянных аргументов.

Листинг 25.25. Файл ru_draw_trass_text_ltype.lsp

```
(defun START (txt arc_enabled len_dash first_point second_point)
  (ru-trass-draw-lw-txt txt len_dash first_point second_point
    arc_enabled t (ru-lw-current) 0)
  (princ)
)
```

Вызов этой программы также включается в XML-меню, в макросы которого записаны подготовленные вызовы программы (листинг 25.26).

Листинг 25.26. Фрагмент XML-меню трасс

```
<item name='Теплосеть линия' image='TOPO\SETI\T_T_S.GIF'
 comment='Рисование трассы любой тепловой сети типом линии с текстовым обозначением'
macro='(progn(if(ru-app-load "ru_draw_trass_text_ltype")(start "T" NIL NIL NIL
NIL)))'/>
<item name='Теплопровод подающий со спутником' image='TOPO\SETI\TR_102_1.GIF'
comment='Рисование трассы типом линии с текстовым обозначением'
macro='(progn(ru-app-load "ru_draw_trass_text_ltype")(start "T102.1" NIL NIL NIL
NIL))'/>
```

Такое меню является одновременно и справочником обозначений, т. к. далеко не каждый пользователь помнит, для каких трубопроводов применяется обозначение T102.1.

А для рисования линии с любым обозначением пригодится программа, приведенная в листинге 25.27.

Листинг 25.27. Файл ru_draw_trass_txt_line.lsp

```
(defun START (/ txt)
 (while (setq txt (ru-get-string "Обозначение линии: " "T1"))
  (ru-trass-draw-lw-txt txt nil nil nil T T (ru-lw-current) 0))
 (princ))
(START)
```

Врезка и привязки текстов к линиям

Программы-близнецы для врезки текста в линию (листинг 25.28) и для написания текста над линией (листинг 25.29) пригодятся, когда использование текстового типа линии неприемлемо. Напомним, что используемые в них функции всегда пишут строку с правильной ориентацией относительно линии — текст не может оказаться написанным "вверх ногами".

Листинг 25.28. Файл ru_txt_in_line.lsp

```
(defun START (/ txt)
  (ru-app-begin) (setq txt "")
  (while(setq txt (ru-get-string "Текст для врезки в линию" txt))
     (ru-draw-txt-in-line txt))
  (ru-app-end)(princ))
(START)
```

Листинг 25.29. Файл ru_txt_along_line.lsp

```
(defun START (/ txt)
  (ru-app-begin) (setq txt "")
  (while (setq txt (ru-get-string  "Текст для написания НАД линией" txt))
     (ru-draw-txt-up-line txt))
  (ru-app-end) (princ))
(START)
```

Несложно написать и модифицированные программы, врезающие предопределенные в XML-меню тексты в линии.

Рисование контуров

Топографам и "генпланистам" часто приходится рисовать замкнутые контуры с определенным типом линии, при этом контур может быть сглаженным. Решает такие задачи программа, показанная в листинге 25.30.

Листинг 25.30. Файл ru_draw_dot_closed_pline.lsp

```
(defun START (ltype / old_ltype)
(ru-app-begin)
```

```
(setq old_ltype (getvar "CELTYPE")) (ru-ltype-set-current ltype)
 (while (ru-pline-draw-closed-lw-msg "замкнутого контура" t
     (ru-lw-current) 0)
     (if (ru-yes " Сгладить контур")
      (vl-cmdf "_.PEDIT" (ssadd (entlast) (ssadd)) "_FIT" "_X")))
  (ru-ltype-set-current old_ltype)
  (ru-app-end)
  (princ)
)
```

Вызов также производится посредством XML-меню (листинг 25.31).

Листинг 25.31. Фрагмент XML-меню контуров

```
<item name='Контур пунктирный' image='DRAW\PUNKTIR.GIF'
 comment='Замкнутый пунктирный контур'
 macro='(progn(if(ru-app-load "ru_draw_dot_closed_pline")(START "HIDDEN")))'/>
  <item name='Контур точечный' image='DRAW\DOTLINE.GIF'
 comment='Замкнутый точечный контур'
macro='(progn (if(ru-app-load "ru_draw_dot_closed_pline")(START "DOT")))'/>
```

Разумеется, существуют и варианты рисования незамкнутых линий заданного типа со сглаживанием по запросу.

Рисование прямоугольников

У нас имеется много программ для рисования различных изображений, но разберем мы только одну — рисование прямоугольника. Как выглядит прямоугольник, представляют все читатели (в отличие, например, от изображения вентилятора), и такая программа будет понятна всем.

В строительном проектировании очень часто приходится рисовать различные объекты прямоугольной формы. Стандартная команда RECTANG (ПРЯМОУГ) не очень удобна — хотя бы из-за необходимости поворота системы координат для рисования под углом. Написать свою программу рисования прямоугольника можно и в две строки, но пользы от такой программы будет мало. Мы разработаем довольно "хитрую" программу, которая пригодится многим специалистам.

В диалоговом окне **Прямоугольник** (рис. 25.4) этой программы можно:

- задать размеры сторон прямоугольника вводом с клавиатуры или измерением в рисунке (размеры сторон запоминаются и восстанавливаются);
- размеры сторон можно указать во время работы;
- возможны три способа рисования — по стороне, по углу и по центру;
- возможна установка веса линии прямоугольника.

Кроме того, имеются и скрытые особенности, не просматривающиеся в диалоговом окне. На рис. 25.5 показаны некоторые изображения, созданные программой рисования прямоугольников.

На самом деле программа не рисует прямоугольники отрезками или полилиниями, а создает блоки прямоугольных объектов. Частным случаем является обычный прямо-

угольник. Кроме линейного контура прямоугольника или вместо контура заполнителем прямоугольника может являться другой блок, лучше единичный.

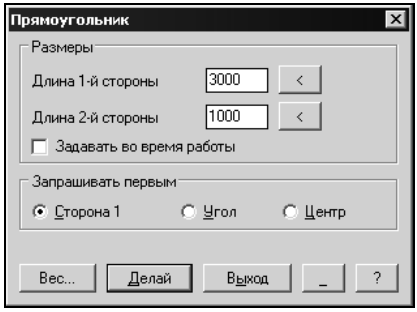


Рис. 25.4. Диалоговое окно рисования прямоугольника

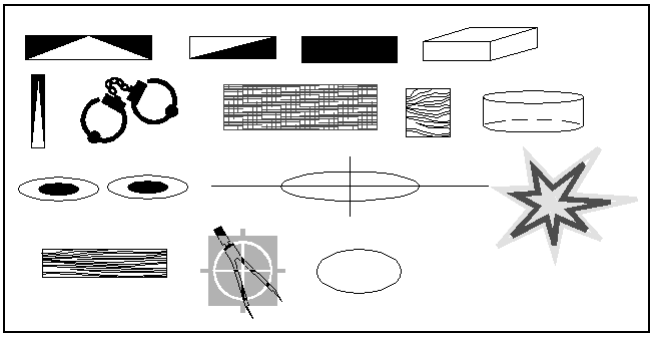


Рис. 25.5. Все это "прямоугольники"

Зачем мы создаем из контура блок? Да потому, что одинаковых прямоугольников в рисунке может быть очень много, например здания на плане города. Если прямоугольный объект создан в виде блока, то его удобно повторять программой "Такой же" (см. листинг 25.7). Прием создания блоков мы применяем довольно часто, например, для рисования различных окружностей — разных диаметров, с разным весом линии, с центровыми линиями или без них.

Функции `ru-draw-rectangle` (листинг 25.32) передаются аргументы:

- `dlg_caption` — заголовок диалогового окна, если `nil` — принимается "Прямоугольник";
- `block_lib` — имя библиотеки блоков, если `nil` — принимается блок из имеющихся в рисунке;
- `block_name_unit` — имя добавочного ЕДИНИЧНОГО блока, если `nil` — создается пустой прямоугольник;
- `hole` — флаг `T`|`nil` необходимости рисования люков.

Последний аргумент появился после того, как выяснилось, что программа часто применяется для рисования резервуаров или камер тепловых сетей с неизвестным заранее количеством люков.

Листинг 25.32. Функция ru-draw-rectangle

```
(defun ru-draw-rectangle (dlg_caption block_lib block_name_unit hole /
  dcl_id do is_ask_dim is_by_angle is_by_center is_by_side lst_x_y
  what_next x y _test _rect_set_as _rect-by-center _rect-by-angle
  _rect-by-side _rect-block _rect-draw-hole)
;|-------------------------------------------------------------------------
Функция рисования люков.
-------------------------------------------------------------------------|;
 (defun _rect-draw-hole (lineweight / diam_hole pnt)
  (setq diam_hole (ru-conv-millimeter-to-unit 600.0))
   (while (setq pnt (ru-get-point-or-exit "Центр люка" nil))
             (ru-draw-circle-and-cross pnt diam_hole 0 lineweight))
   )
;|-------------------------------------------------------------------------
Функция создания прямоугольного блока из единичного блока.
Аргументы:
block_name_unit - имя дополнительного единичного блока;
ent - примитив контура прямоугольника;
t0 - базовая точка для создания блока;
tv - точка вставки блока;
x - масштаб по оси x;
y  - масштаб по оси y;
ang - угол поворота;
lineweight - вес линии;
-------------------------------------------------------------------------|;
 (defun _rect-block (block_name_unit ent t0 tv x y ang lineweight /
  prefix block_name selection)
 (if block_name_unit
    (setq prefix block_name_unit) (setq prefix "RU_BOX"))
  (setq  block_name (strcat prefix "_" (itoa lineweight) "-"
  (itoa (fix (* x 1000))) "-" (itoa (fix (* y 1000)))))
 (if (tblsearch "BLOCK" block_name)
   (progn (entdel ent) (ru-block-insert-obj block_name tv 1 1 1 ang))
   (progn
   (setq selection (ssadd) selection (ssadd ent selection))
   (if block_name_unit
     (progn
       (ru-block-insert-obj block_name_unit t0 x y 1 0)
       (setq selection (ssadd (entlast) selection))
     ))
   (ru-block-din-make block_name t0 selection lineweight)
   (if (tblsearch "BLOCK" block_name)
       (ru-block-insert-obj block_name tv 1 1 1 ang)
       (alert (strcat "Не создан блок " block_name))
   )
  )
 )
  (setq selection nil selection (ssadd)
   selection (ssadd (entlast) selection))
 selection
)
```

```
;|-------------------------------------------------------------------------
Рисование прямоугольника по стороне
-------------------------------------------------------------------------|;
(defun _rect-by-side (block_name_unit x y is_ask_dim hole / ang om pt0 pt1 pt2 pt3
pt4 sa)
(setq om (getvar "ORTHOMODE") sa (getvar "SNAPANG"))
 (while (setq pt1 (ru-get-point-or-exit "Начало 1-й стороны" nil))
   (setq pt2 (ru-get-point-reguired
        (if is_ask_dim "Конец 1-й стороны" "Направление 1-й стороны")
                 pt1) ang (angle pt1 pt2))
 (if is_ask_dim (setq x (distance pt1 pt2)) (setq pt2 (polar pt1 ang x)))
 (if is_ask_dim (progn
   (grdraw (trans pt1 0 1) (trans pt2 0 1) -1)
   (setvar "ORTHOMODE" 1) (setvar "SNAPANG" ang)
   (setq y (ru-get-dist "Длина второй стороны" y pt1))
   (grdraw (trans pt1 0 1) (trans pt2 0 1) -1)))
;;рисуем под углом 0 для создания блока
 (setq pt0 (getvar "VSMAX") pt2 (polar pt0 0 x)
    pt4 (polar pt0 (ru-geom-go-left 0) y)
     pt3 (polar pt2 (ru-geom-go-left 0) y))
 (ru-pline-add (list pt0 pt2 pt3 pt4) t 0 (ru-lw-current) nil)
 (setvar "ORTHOMODE" om) (setvar "SNAPANG" sa)
 (_rect-block block_name_unit (entlast) pt0 pt1 x y ang (ru-lw-current))
 (if hole (_rect-draw-hole (ru-lw-current)))
);_ end of while
 (list x y)
)
;|-------------------------------------------------------------------------
Рисование прямоугольника по углу
-------------------------------------------------------------------------|;
(defun _rect-by-angle (block_name_unit x y is_ask_dim hole / pt0 pt1 pt2 pt3 pt4
selection)
 (while (setq pt1 (ru-get-point-or-exit "Левый нижний угол" nil))
  (if is_ask_dim
   (setq x (ru-get-dist "Длина 1-й стороны" x pt1)
         y (ru-get-dist "Длина 2-й стороны" y pt1)))
 (setq pt0 (getvar "VSMAX") pt2 (polar pt0 0 x)
      pt4 (polar pt0 (ru-geom-go-left 0) y)
      pt3 (polar pt2 (ru-geom-go-left 0) y))
 (ru-pline-add (list pt0 pt2 pt3 pt4) t 0 (ru-lw-current) nil)
 (if (setq selection (_rect-block block_name_unit (entlast) pt0 pt1 x y
                           0 (ru-lw-current)))
  (progn
   (prompt "\nУгол поворота: ")
    (vl-cmdf "_.ROTATE" selection "" (trans pt1 0 1))
     (while (> (getvar "CMDACTIVE") 0) (vl-cmdf pause))
 )
)
(if hole (_rect-draw-hole (ru-lw-current)))
)
 (list x y)
)
```

```
;|-------------------------------------------------------------------------
Рисование прямоугольника по центру
-------------------------------------------------------------------------|;
(defun _rect-by-center (block_name_unit x y is_ask_dim hole / pt0 pt1 pt2 pt3 pt4
selection)
 (while (setq pt1 (ru-get-point-or-exit "Центр" nil))
  (if is_ask_dim (setq
       x (* (ru-get-dist "Полудлина 1-й стороны" (/ x 2) pt1) 2)
       y (* (ru-get-dist "Полудлина 2-й стороны" (/ y 2) pt1) 2)))
  (setq pt0 (getvar "VSMAX") pt2 (polar pt0 0 x)
            pt4 (polar pt0 (ru-geom-go-left 0) y)
            pt3 (polar pt2 (ru-geom-go-left 0) y))
  (ru-pline-add (list pt0 pt2 pt3 pt4) t 0 (ru-lw-current) nil)
  (setq pt2 (polar pt1 (ru-geom-go-back 0) (/ x 2))
        pt2 (polar pt2 (ru-geom-go-right 0) (/ y 2)))
  (if (setq selection
   (_rect-block block_name_unit (entlast) pt0 pt2 x y 0 (ru-lw-current)))
    (progn
     (prompt "\nУгол поворота: ")
     (vl-cmdf "_.ROTATE" selection "" (trans pt1 0 1))
     (while (> (getvar "CMDACTIVE") 0) (vl-cmdf pause))))
  (if hole (_rect-draw-hole (ru-lw-current)))
);_ end of while
(list x y)
)
;|-------------------------------------------------------------------------
Далее идут простые функции для работы с диалоговым окном. Обычно мы не стремимся их
"по-Лисповски" оптимизировать, т. к. при этом легко потерять логику работы.
-------------------------------------------------------------------------|;
 (defun _rect_set_as ()
 (if is_by_side
  (progn (set_tile "as_side" "1") (set_tile "as_angle" "0")
       (set_tile "as_centr" "0")))
 (if is_by_angle
  (progn (set_tile "as_side" "0") (set_tile "as_angle" "1")
                (set_tile "as_centr" "0")))
 (if is_by_center
  (progn (set_tile "as_side" "0") (set_tile "as_angle" "0")
         (set_tile "as_centr" "1")))
)
(defun _test ()
  (mode_tile "button_do" 0)
  (setq is_ask_dim (= (get_tile "in_work") "1"))
  (if is_ask_dim
    (progn (set_tile "in_work" "1") (mode_tile "ed_y" 1)
      (mode_tile "get_y" 1) (mode_tile "ed_x" 1) (mode_tile "get_x" 1))
    (progn (set_tile "in_work" "0") (mode_tile "ed_y" 0)
      (mode_tile "get_y" 0) (mode_tile "ed_x" 0) (mode_tile "get_x" 0))
 )
  (set_tile "error" "")
  (setq x (atof (get_tile "ed_x")) y (atof (get_tile "ed_y")))
```

```
  (if (<= x 0) (set_tile "in_work" "1"))
  (if (<= y 0) (set_tile "in_work" "1"))
  (setq is_ask_dim (= (get_tile "in_work") "1")
              is_by_side   (= (get_tile "as_side") "1")
              is_by_angle  (= (get_tile "as_angle") "1")
              is_by_center (= (get_tile "as_centr") "1"))
)
;|-----------------------------------------------------------------------
Основная функция
-----------------------------------------------------------------------|;
 (ru-app-begin)
 (if (null dlg_caption) (setq dlg_caption "Прямоугольник"))
 (setq Do T)
 (if (and block_lib block_name_unit)
     (setq Do (ru-block-lib-insert block_lib block_name_unit)))
 (if Do
  (progn
    (setq X (atof (ru-user-read-last-param "RectangX" "1.0"))
       Y (atof (ru-user-read-last-param "RectangY" "1.0"))
       is_ask_dim T is_by_side T is_by_angle  nil is_by_center nil
       what_next 5)
    (if (setq dcl_id (load_dialog (ru-file-dcl "ru_draw_rectangle")))
      (progn
       (while (> what_next 0)
        (if (new_dialog "rectang" dcl_id)
          (progn
            (set_tile "title" dlg_caption) (set_tile "ed_x" (rtos x))
            (set_tile "ed_y" (rtos y)) (_rect_set_as) (_test)
            (action_tile "as_side" "(_test)")
            (action_tile "as_angle" "(_test)")
            (action_tile "as_centr" "(_test)")
             (action_tile "ed_x" "(_test)")
             (action_tile "ed_y" "(_test)")
             (action_tile "in_work" "(_test)")
             (action_tile "get_x" "(_test)(done_dialog 2)")
             (action_tile "get_y" "(_test)(done_dialog 3)")
             (action_tile "lineweight" "(ru-dlg-dcl-select-lw)")
             (action_tile "button_do" "(_test)(done_dialog 4)")
             (action_tile "help" "(ru-dlg-view-txt-file
              (ru-file-help-txt \"ru_draw_rectangle.txt\") \"О программе\"
\"Рисование прямоугольника\" (and *ru_developer* (ru-user-may-txt-hlp-edit)))")
            (setq what_next (start_dialog))
            (cond
              ((= 0 what_next))
              ((= 2 what_next)
               (setq X (ru-get-dist "Длина 1-й стороны" X nil)))
              ((= 3 what_next)
               (setq Y (ru-get-dist "Длина 2-й стороны" y nil)))
              ((= 4 what_next)
               (if is_by_side
                  (setq lst_x_y
                    (_rect-by-side block_name_unit x y is_ask_dim hole)
```

```
                    x (car lst_x_y) y (cadr lst_x_y)
                    is_by_angle NIL is_by_center NIL))
                (if is_by_angle
                   (setq lst_x_y
                    (_rect-by-angle block_name_unit x y is_ask_dim hole)
                    x (car lst_x_y) y (cadr lst_x_y)
                    is_by_side nil is_by_center nil))
                (if is_by_center
                   (setq lst_x_y
                    (_rect-by-center block_name_unit x y is_ask_dim hole)
                    x (car lst_x_y) y (cadr lst_x_y)
                    is_by_side nil is_by_angle nil))
              )
               ((= 8 what_next)
               ;; определено в базовом DCL
                 (ru-dcl-hide-dialog)))
     )
      (progn (setq what_next 0) (alert "Не могу показать диалоговое окно"))
    )
   )
   (ru-user-write-last-param "RectangX" (rtos x 2 2))
   (ru-user-write-last-param "RectangY" (rtos y 2 2))
   (unload_dialog DCL_ID))
   (alert "Не могу загрузить диалоговое окно")
  )))
 (ru-app-end)
 (princ)
)
```

Приводить исходный текст DCL-файла мы не будем — слишком много он займет места. Вызов функции `ru-draw-rectangle` с различными параметрами из XML-меню позволяет "создать" множество прикладных команд. Часть изображений, показанных в рис. 25.5 (таких, у которых нет окружающего прямоугольного контура), создана с помощью простой функции, приведенной в листинге 25.33. Она позволяет рисовать псевдопрямоугольные объекты неопределенных заранее размеров — таких изображений требуется довольно много.

Листинг 25.33. Функция `ru-draw-rectangle-block-lib-box`

```
(defun ru-draw-rectangle-block-lib-box (block_lib block_name msg /
   ins_pnt second_pnt)
 (ru-app-begin)
 (if (ru-block-lib-insert block_lib block_name)
  (while
    (setq ins_pnt (ru-get-point-or-exit (strcat "Угол " msg) nil))
    (setq second_pnt (ru-get-corner-reguired
      (strcat "Противоположный угол" msg) ins_pnt))
    (ru-block-insert-obj block_name ins_pnt
     (- (car second_pnt) (car ins_pnt))
     (- (cadr second_pnt) (cadr ins_pnt)) 1 0)
))
 (ru-app-end)
)
```

Рисование текстов различными способами

Тексты приходится писать очень часто. Программное создание текстов мы освоили, теперь напишем простые функции, заменяющие команду DTEXT (ДТЕКСТ). Простейшая программа (листинг 25.34) ничего особенного не делает — просто запускает команду DTEXT (ДТЕКСТ), но дает большую экономию времени за счет частоты применения, т. к. пользователь не тратит времени на ввод или подтверждение высоты и угла поворота текста. Для реже встречающихся повернутых текстов предусмотрен флаг запроса угла поворота.

Листинг 25.34. Файл ru_draw_txt_horisontal

```
(defun START (is_horisontal / pnt)
  (ru-app-begin)
  (if (setq pnt (ru-get-point-or-exit "Точка начала текста" nil))
    (vl-cmdf "_.DTEXT" pnt (ru-normal-text-height)
     (if is_horisontal 0 (ru-conv-rad-to-deg
      (ru-get-angle "Угол поворота текста: " 0 pnt)))))
  (ru-app-end)
  (princ)
)
```

Вызов такой команды просто обязан быть в стандартной панели инструментов рисования. Часто применяемые подчеркнутые тексты заголовков изображений удобно рисовать с помощью программы, приведенной в листинге 25.35.

Листинг 25.35. Файл ru_draw_text_underlined.lsp

```
(defun START (/ txt point)
  (ru-app-begin)
  (setq txt (ru-user-read-last-param "LastUserString" "ПЛАН"))
  (while(setq txt (ru-get-string "Подчеркнутый текст " txt))
   (while (setq point
     (ru-get-point-or-exit (strcat "Центр текста '" txt "'") nil))
     (ru-text-add (strcat "%%u" txt) point
        (ru-normal-text-height) 0 acalignmentcenter))
   (ru-user-write-last-param "LastUserString" txt))
  (ru-app-end)
  (princ)
)
(START)
```

Создание и выбор типовых текстов

Во всех разделах проекта используется множество типовых текстов (примечания, общие указания и т. п.). Типовые фразы хранятся в словаре и доступны при вводе текста с использованием функции `ru-get-string`. Типовые тексты удобнее хранить в

файлах, выбирать их из древовидного меню, редактировать перед вставкой и вставлять в нужное место рисунка. Работа с такой программой может выглядеть так:

- производится выбор файла из каталога типовых текстов (рис. 25.6);
- во время выбора файла возможно его редактирование;
- после выбора файла производится вставка текста.

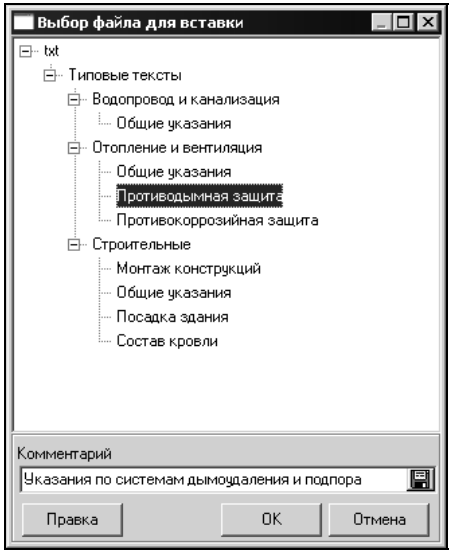


Рис. 25.6. Выбор типового текста для вставки

При вставке текста возможно, в момент запроса точки вставки, выбрать опцию указания количества строк. В этом случае будет написан не весь текст, а указанное количество строк. Это позволит длинный файл написать в несколько колонок. Возможен даже вариант запроса точки для каждой строки, для этого достаточно при вводе количества строк указать единицу. Такой вариант удобен, если, например, в файле находятся строки с типовыми наименованиями помещений. Сначала напишем главную функцию с выбором файла (листинг 25.36).

Листинг 25.36. Функция `ru-text-file-import`

```
(defun ru-text-file-import (/ file_name)
(if (setq file_name (ru-dlg-file-select-in-tree "Выбор файла для вставки"
   (strcat (ru-dirs-get-all-users) "txt\\") ".txt" t))
  (ru-text-draw-file file_name))
(princ)
)
```

Непосредственное написание текстов вынесем в отдельную функцию (листинг 25.37) — она пригодится для вставки любых текстовых файлов.

Листинг 25.37. Функция `ru-text-draw-file`

```
(defun ru-text-draw-file (file_name / pnt start_lst string_list str_count _get-
point-text)
;; Локальная функция ввода точки начала текста
 (defun _get-point-text (msg str_count / point done)
  (while (not done)
   (setq point (ru-get-point-or-exit (strcat msg " Пишем "
      (ru-string-number-end str_count "стро" "ку" "ки" "к")) "Число"))
   (cond
    ((= point "Число")
     (setq str_count (ru-get-int-or-pick "Количество строк " str_count)))
;; Выход, если введена точка или отказались от ввода
    (T (setq done (listp point)))
  )
 )
  (list point str_count)
)
;; Главная функция
 (setq pnt T)
 (if (setq string_list (ru-list-read-from-file file_name))
  (progn
   (setq str_count (length string_list) start_lst (list pnt str_count))
   (while (and pnt string_list
    (setq start_lst (_get-point-text (strcat "Строка '"
   (car string_list) "'\nНачало колонки, осталось "
   (ru-string-number-end (length string_list) " стро" "ка" "ки" "к") ".")
   str_count)))
   (if (setq pnt (car start_lst))
    (repeat (min (setq str_count (cadr start_lst)) (length string_list))
     (ru-text-add (car string_list) pnt (ru-normal-text-height) 0 nil)
     (setq pnt (ru-geom-txt-down-line pnt) string_list (cdr string_list))
 )))))
)
```

Конструктор таблиц

О рисовании и заполнении различных таблиц текстами мы уже писали в *главе 16*. Функция `ru-table-draw-with-ask` (см. листинг 16.20) позволяет нарисовать и разграфить любую таблицу с шапкой. Шапки таблиц у нас хранятся в файлах в каталоге %ruCADRootDir%\All Users\table\. Размеры граф таблицы записаны в файле имя_таблицы.ini. Файл блока таблицы и INI-файл можно создать вручную, а можно и с помощью специальной программы, которую мы обещали предъявить для обозрения, что и делаем. Для облегчения рисования произвольных таблиц мы напишем специальную программу (листинг 25.38). Хотя в системе имеется почти сотня стандартных таблиц, часто возникает необходимость просто разграфить таблицу для разных целей. Отличие программы от простого рисования в том, что высота строк рассчитывается автоматически под нормальную высоту текста.

Листинг 25.38. Файл ru_draw_table_user.lsp

```
(defun START (/ column_width row_count string_height tmp_point first_point
second_point table_height column_number)
 (ru-app-begin)
 (while (setq first_point
   (ru-get-point-or-exit "Левый верхний угол граф" nil))
    (setq tmp_point first_point string_height
           (ru-get-dist (strcat "Высота строки, " (ru-unit-name))
             (ru-normal-table-row-height) first_point)
          table_height
           (_ru-get-with-default (strcat "Примерная высота таблицы, "
                     (ru-unit-name)) (rtos string_height) 'getdist 4
             "Строк" first_point)
    )
     (cond
       ((= table_height "Строк")
        (setq row_count
             (ru-get-int-or-pick "Количество строк в таблице" 1)
              table_height (* row_count string_height))
       ((null table_height)
        (setq table_height string_height row_count 1))
       (T
        (setq row_count
         (ru-match-ceiling (/ table_height string_height))
              table_height (* row_count string_height)))
    )
     (princ (strcat "\nБудет начерчено " (itoa row_count) " строк! "))
     (setq column_width  (* 2 string_height) column_number 0)
    ;; Вертикальные линиии
     (while
       (setq column_width (ru-get-dist-or-exit
                (strcat "Ширина графы " (itoa (1+ column_number)))
                column_width first_point))
        (if (= column_number 0)
          (ru-line-add first_point
            (polar first_point (ru-geom-go-right 0) table_height) 0 nil)
       )
        (setq first_point   (polar first_point 0 column_width)
              column_number (1+ column_number))
        (ru-line-add first_point
          (polar first_point (ru-geom-go-right 0) table_height) 0 nil)
    )
     (if (> column_number 0)
       (progn
         (ru-line-add tmp_point first_point 0 nil)
         (repeat row_count
           (setq first_point
             (polar first_point (ru-geom-go-right 0) string_height)
                 second_point
                    (polar tmp_point (ru-geom-go-right 0) string_height)
```

```
                tmp_point second_point)
            (ru-line-add  second_point  first_point 0 nil)
        )))))  (ru-app-end) (princ))
(START)
```

В результате может быть нарисована таблица, подобная показанной на рис. 25.7

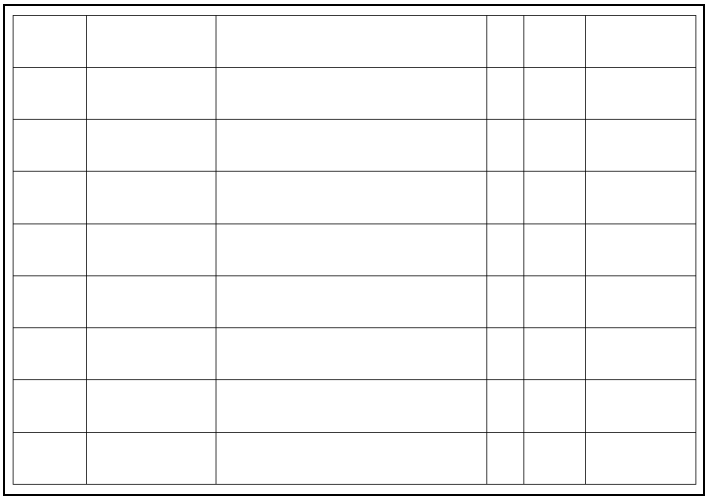


Рис. 25.7. Результат рисования произвольной таблицы

Вызов программы ru_draw_table_user мы смело можем включать в XML-меню таблиц.

```
<item name='Разграфить' image='DRAW\RAZGRAF.GIF'
comment='Разграфка в виде сетки для таблицы'
macro='(ru-app-load "ru_draw_table_user")'/>
```

Теперь займемся созданием стандартной таблицы, шапка которой будет размещена в каталоге таблиц. Шапку, включая заголовки колонок и таблицы, надо предварительно нарисовать любыми средствами. Разумеется, при этом лучше использовать наши средства, поддерживающие пропорциональность символов. Создается стандартная таблица функцией, показанной в листинге 25.39.

Листинг 25.39. Файл ru_pro_table_gen.lsp

```
(defun START (/ block_header bottom_left_pnt description dim_string do file_name
header_height table_ini table_lst tmp_dist tmp_pnt top_left_pnt vla_array_objs ss)
(ru-app-begin)
 (if
  (ru-yes (strcat
   "Программа записывает таблицу"
   "\n и генерирует для нее файл описания!"
  "\nШапка должна быть предварительно нарисована!\nБудем делать"))
```

```
;;; Ввод в двухстрочном диалоговом окне имени и описания
  (if (setq table_lst (ru-dlg-get-two-string "Описание таблицы" "Файл"
      "ru_new_table" T "Описание" "Новая таблица" T T "" ""))
   (progn
    (setq block_header (car table_lst) description  (cadr table_lst)
          file_name (ru-file-set-ext (ru-file-table block_header) ".dwg")
          do (if (findfile file_name)
          (not (ru-no
           (strcat "Файл " file_name " уже имеется. \nПереписать"))) T)
   )
    (if do (progn
      (setq top_left_pnt
           (ru-get-point-reguired "Левый верхний угол шапки" nil)
            bottom_left_pnt
           (ru-get-point-reguired "Левый нижний угол шапки" top_left_pnt)
            header_height   (distance top_left_pnt bottom_left_pnt)
            tmp_pnt bottom_left_pnt tmp_dist header_height dim_string "")
      (while (setq tmp_dist
                   (ru-get-dist-or-exit "Длина графы" tmp_dist tmp_pnt))
;; Формируем строку размеров в миллиметрах на бумаге
       (setq dim_string (strcat dim_string
           (rtos (ru-conv-unit-to-millimeter-in-paper tmp_dist) 2 2) ";")
           tmp_pnt (polar tmp_pnt 0 tmp_dist))
     )
      (princ "\nВыбери примитивы, входящие в шапку: ")
      (setq ss (ru-ss-get))
;; Масштабируем набор в миллиметры на бумаге
      (ru-obj-ent-ss-scale ss top_left_pnt
           (ru-conv-unit-to-millimeter-in-paper 1.0))
;; Для удобства модификации создаем массив объектов
      (setq vla_array_objs (vlax-make-variant (ru-ss-to-vla-array ss)))
;; Устанавливаем стандартные свойства блока
      (ru-obj-vla-array-mod vla_array_objs "Layer" "0")
      (ru-obj-vla-array-mod vla_array_objs "Color" acbyblock)
;; Создаем блок в рисунке
      (if (ru-block-obj-make-from-vla-array block_header top_left_pnt
          vla_array_objs)
       (progn
;; А записать лучше командой
        (if (findfile file_name)
         (command "_.WBLOCK" file_name "_Yes" block_header)
         (command "_.WBLOCK" file_name block_header)
       )
;; Создаем INI-файл с описанием таблицы
        (setq table_ini (ru-file-set-ext file_name ".ini"))
        (ru-ini-write table_ini "Table" "table_name" description)
        (ru-ini-write table_ini "Table" "header_block" block_header)
        (ru-ini-write table_ini "Table" "header_height"
         (rtos (ru-conv-unit-to-millimeter-in-paper header_height) 2 2))
        (ru-ini-write table_ini "Table" "columns_width"
```

```
        (ru-string-rem-right dim_string (list ";")))
       (ru-file-write-dirinfo-file-comment file_name  description)
     ))))))) (ru-app-end) (princ))
(START)
```

Вставку таблиц с разграфкой мы рассмотрели в *главе 16*.

Замечание

Наши средства для рисования таблиц не стоит сравнивать с инструментами системы AutoCAD 2005 или программой ATable Александра Щетинина — это различные классы программ. Так же, как бензиновые или электрические пилы не вытеснили простые ножовки, так и наряду со сложными программами нужны простые инструменты.

Заполнение таблиц

Вариант заполнения таблиц с редактированием мы рассматривали в *главе 16*. Сейчас ознакомимся еще с одной полезной программой — заполнением ведомости ссылочных и прилагаемых документов. В эту ведомость, являющуюся обязательным компонентом общих данных по рабочим чертежам, записывают использованные в проекте документы — обычно это типовые серии рабочих чертежей изделий. В некоторых разделах количество используемых ссылочных документов достигает нескольких десятков. Для каждого ссылочного документа необходимо записать точное обозначение и наименование. Эту рутинную работу, занимающую значительное время, можно легко автоматизировать (листинг 25.40). Легко, потому что мы располагаем большой библиотекой функций, в которых уже решены все частные задачи. Работа начинается с отбора документов в XML-меню (рис. 25.8).

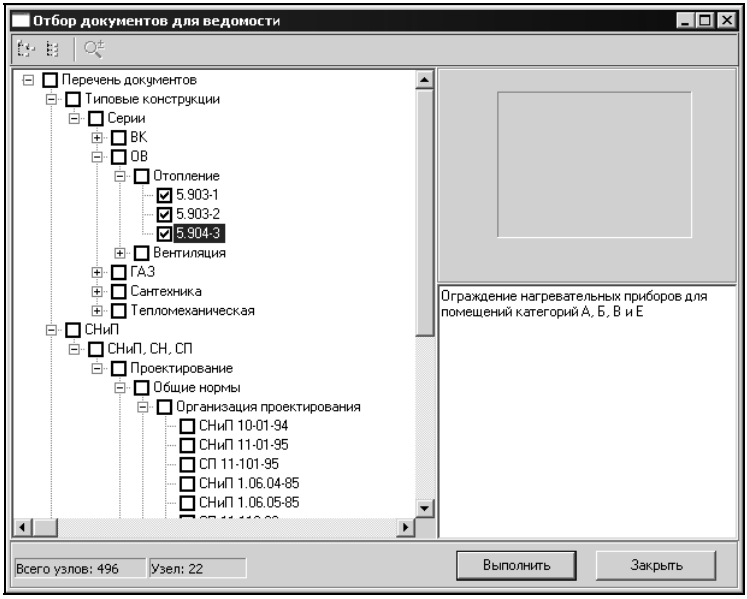


Рис. 25.8. Отбор документов из XML-меню

После выбора документов запрашивается точка начала текстов (таблица ведомости должна быть нарисована) и ведомость заполняется.

Листинг 25.40. Файл ru_draw_table_docs.lsp

```
(defun START (/ doc_list header pnt tmp)
  (ru-app-begin)
  (if (setq doc_list
        (_ru-xml-tree-select "Отбор документов для ведомости"
        (ru-file-menu-xml "snip\\docs.xml") (list "name" "doc_name")
         "name" "doc_name" "doc_name" T nil nil))
   (progn
     (if (setq header (ru-get-string "Заголовок списка документов"
               "ССЫЛОЧНЫЕ ДОКУМЕНТЫ"))
       (progn
         (setq pnt (polar (ru-get-point-reguired
                     "Левый нижний угол графы первой строки" nil)
                    (ru-geom-go-left 0)
                    (ru-conv-millimeter-in-paper-to-unit 2.0)))
         (if (/= header "")
            (progn
              (ru-text-add (strcat "%%U" header)
                (polar pnt 0 (ru-conv-millimeter-in-paper-to-unit 107.0))
                (ru-normal-text-height) 0.0 acAlignmentCenter)
              (setq pnt (ru-geom-txt-down-line pnt))))
       ))
     (foreach doc doc_list
;; Уточняем начало строки, может быть нужно начать новую колонку
       (if (setq tmp (_ru-get-with-default
                       "Левый нижний угол графы следующей записи"
                       "Авто" 'getpoint nil nil nil))
          (setq pnt (polar (trans tmp 1 0) (ru-geom-go-left 0)
                      (ru-conv-millimeter-in-paper-to-unit 2.0))))
       (setq pnt (ru-text-draw-in-table pnt
                   (list (ru-get-from-xml-assoc "name" doc)
                         (ru-get-from-xml-assoc "doc_name" doc))
                   (list 60.0 95.0)))
     );_ end of foreach
   ))(ru-app-end) (princ))
(START)
```

Результат работы программы показан на рис. 25.9.

Эта же программа используется для заполнения других ведомостей с такими же размерами граф. Кроме стандартных форм, например ведомости основных комплектов рабочих чертежей, мы рекомендуем пользователям создавать ведомость нормативных и исходных документов. Солидная таблица ссылок на конкретные нормативы, обозначения технических условий, заданий, технологических планировок выглядит лучше, чем невнятные фразы в общих указаниях. Да и главный инженер проекта, подписывая запись о соответствии проекта действующим нормам и правилам (она

также создается специальной программой), будет видеть, каким конкретно нормам соответствует проект. В *случае чего* он не сможет уходить от своей доли ответственности, ссылаясь на незнание.

ВЕДОМОСТЬ ССЫЛОЧНЫХ И ПРИЛАГАЕМЫХ ДОКУМЕНТОВ

Обозначение	Наименование	Примечание
	ССЫЛОЧНЫЕ ДОКУМЕНТЫ	
3.900-9 вып.2	Опорные конструкции и средства	
	крепления трубопроводов к	
	металлическим колоннам	
3.900-9 вып.1	Опорные конструкции и средства	
	крепления трубопроводов к	
	железобетонным колоннам	
3.900-9	Опорные конструкции и средства	
	крепления трубопроводов	
5.904-3	Ограждение нагревательных приборов	
	для помещений категорий А, Б, В и Е	
5.903-2	Воздухосборники для систем отопления	
	и теплоснабжения вентиляционных	
	установок	

Рис. 25.9. Заполненная ведомость ссылочных и прилагаемых документов

ГЛАВА 26

Формирование специализированных программ из универсальных функций

Мы рассмотрели множество функций разнообразного назначения, но они не являются нашей конечной целью. Конечному пользователю нужны не функции, а кнопки или пункты меню, при щелчке по которым начинают работать наша программа или макроопределение, выполняющие определенные полезные действия. Теперь мы должны превратить наработанные идеи в конкретные решения для пользователей. Примерно пятьдесят конечных программ мы, между делом, уже рассмотрели — названия листингов у программ начинались со слова "Файл", а не "Функция". Это и есть конечная продукция, включаемая в меню. Как правило, мы используем XML-меню и только самые ходовые программы привязываем к кнопкам панелей инструментов[1]. В этой главе мы рассмотрим приемы превращения функций, полезных для программиста, в программы, полезные для пользователя.

Эффективные программы длиной в одну строку

Напишем простейшую программу (листинг 26.1).

Листинг 26.1. Файл ru_block_insert_lib.lsp

```
(defun START (block_lib_name block_name ins_code x_scale y_scale
              is_angle0)
(ru-block-insert-from-lib
  block_lib_name block_name ins_code x_scale y_scale is_angle0)
(princ)
)
```

Эта мини-программа просто вызывает известную нам функцию, передавая ей все аргументы без изменений. Зачем же это сделано и почему нельзя напрямую вызвать функцию, тем более что аргументы не изменяются?

---

[1] В системе AutoCAD 2005 появилась возможность привязки программ в виде иллюстрированных инструментов на вкладках немодальных окон Tool Palettes.

Причина в том, что *функция* тем самым превращена в *программу*, загружаемую из отдельного файла. При загрузке файла, в отличие от вызова функции из памяти, у нас происходят (или могут происходить) какие-то дополнительные действия, определенные в функции `ru-app-load` — например, может выводиться предварительная справка о программе. О постоянной перегрузке программ мы рассуждали в *главе 10*, и, независимо от того, нравится или нет такой подход читателям, мы используем именно такую технологию, испытанную в течение десятка лет.

Теперь нам достаточно написать в XML-меню сотни или тысячи раз нечто наподобие показанного в листинге 26.2, и пользователи будут иметь множество полезных программ.

Листинг 26.2. Пример XML-меню с функцией вставки библиотечных блоков

```
<?xml version='1.0' encoding='windows-1251'?>
<root name='КИП' macro=''>
  <item name='Прибор по месту'
   image='EO\KIP\_E1_119.GIF'
  comment='Множественная вставка изображения с автомасштабированием'
   macro='(progn (if(ru-app-load "ru_block_insert_lib")
         (start "000000e7" "ПРИБОР_МЕСТН_КРУГ" 1 1 1 NIL)))'/>
  <item name='Прибор на щите'
   image='EO\KIP\_E1_121.GIF' comment='Прибор, устанавливаемый на щите'
   macro='(progn(if (ru-app-load "ru_block_insert_lib")
          (start "000000e7" "ПРИБОР_ЩИТ_КРУГ" 1 1 1 NIL)))'/>
</root>
```

Замечание

В макроопределении мы сделали "обертку" в виде функции `progn`, чтобы все макроопределение было единым выражением, т. к. производится его синтаксический анализ. В конкретном примере функцию `progn` можно было бы исключить (роль "обертки" играет функция `if`), но мы этого не делаем, т. к. нет гарантии, что завтра нам не захочется вписать после `if` вызов функции `princ` или еще что-то.

После наполнения меню пользователь сможет выбирать иллюстрированные программы (рис. 26.1), возможно, не догадываясь, что будет вставлять блоки.

Важно то, что теперь наращивание системы может осуществляться пользователями. Опыт показывает, что "смышленая тетка", обученная работе с текстовым редактором, знающая свою предметную область и умеющая аккуратно рисовать блоки, вполне справится с этой работой. Достаточно ей показать, как это делается, объяснить что можно изменять (имя библиотеки и блока), а что не нужно трогать, и дело пойдет.

> В способность "теток" и пользователей вообще на подобные действия не верят многие программисты, а зря. Надо уметь не только программировать, но и относиться к пользователям без снобизма, говорить с ними на одном языке, объяснять, в чем заключается их личный интерес, и дело пойдет. Вы получите достаточное количество союзников, готовых включиться в такую работу. *(Сергей Зуев)*

Подробно создание XML-меню мы рассматривали в *главе 16*. Напомним только, что они могут быть вложенными и иллюстрации к пунктам меню могут создаваться "на лету".

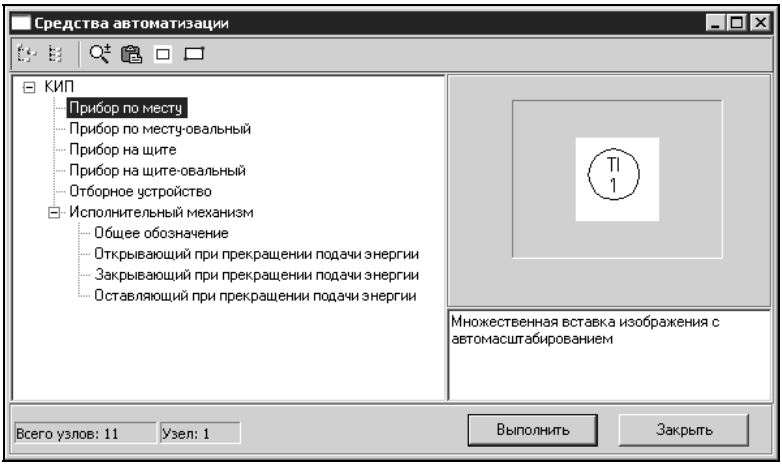


Рис. 26.1. Выбор блока из иллюстрированного меню

Создание "специального" меню

В *главе 8* мы писали о том, что все специализированные программы будем размещать на шестом месте в главном падающем меню **СпецРис**. В этом меню мы, с разбивкой на тематические страницы, соберем команды специального рисования[1]. Каждая тематическая страница вызывается функцией `ru-menu-special-show` (пример вызова приводился в *главе 8*). На каждой тематической странице могут быть пункты меню для вызова групп изображений.

Принципы разделения программ между меню системы AutoCAD (а внутри него между падающими, вложенными меню и панелями инструментов) и XML-меню мы уже обсуждали. Вопросы эти неоднозначные и во многом зависят от личных пристрастий и "натасканности" пользователей на те или иные задачи. В любом случае наша система открыта и квалифицированные пользователи (а лучше администраторы САПР) могут "перетасовать" меню по своему вкусу. На рис. 26.2 показано падающее меню **СпецРис-ЭО**, в котором вся "электрика" разбита на несколько крупных групп изображений.

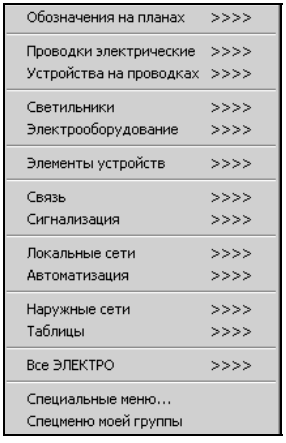


Рис. 26.2. Падающее специальное меню "электрики"

[1] В AutoCAD 2005 мы можем продублировать специальные меню в Tool Palettes.

В числе прочих мы видим и пункт **Автоматизация**, и **Все ЭЛЕКТРО**. При выборе пункта **Автоматизация** будет вызвано XML-меню, показанное на рис. 26.1, а при выборе **Все ЭЛЕКТРО** — полное XML-меню "электрики" (рис. 26.3), в котором, в одной из ветвей, будет и **Автоматизация**. Какой именно пункт применить — решит пользователь. Напомним, что мы вообще можем спрятать все пункты всех меню под одной кнопкой, но пользоваться таким деревом, конечно, будет неудобно.

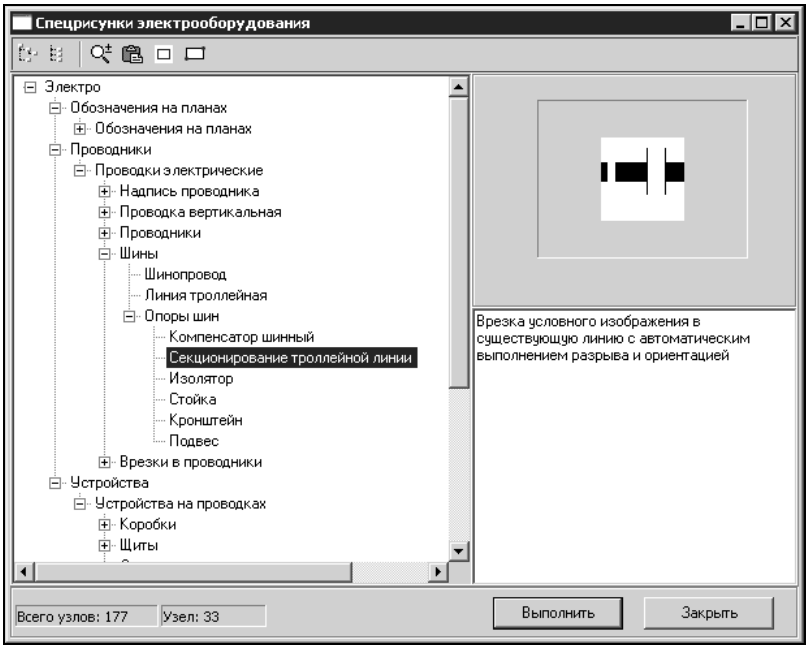


Рис. 26.3. Полное XML-меню электрооборудования

Вызов XML-меню производится из обычного меню ruCAD. Пример вызова специального меню рисования средств автоматизации показан в листинге 26.3.

Листинг 26.3. Фрагмент файла ruCAD.mnu

```
[Автоматизация\t>>>>]^C^C^P(defun C:RU()(ru-xml-pop-mnu "KIP" "Средства
автоматизации")(princ));RU
```

После того, как продумана структура специальных меню, их можно наполнять вызовами XML-меню и формировать сами XML-файлы. На этом этапе очень важно сотрудничество с пользователями — только они смогут оценить удобство работы. Напоминаем, что наши XML-меню имеют очень гибкую структуру и позволяют использовать включения других XML-меню. Если, например, электрикам будет удобнее, чтобы в меню электрооборудования были не только проводники и устройства, а и отметки или линии, то это очень легко реализовать, не отнимая те же отметки у строителей или сантехников.

Группы команд, использующих одну функцию

Рассмотрим самые распространенные группы команд, использующих одни и те же функции в разных разделах проекта. Одну из них мы уже видели (листинг 26.2). Вставка блоков из библиотек блоков с автоматическим масштабированием применяется везде. Для написания таких пунктов меню надо только знать имя блока, имя библиотеки, в которой он "живет", масштабный фактор для вставки блока и возможен ли поворот блока. Вызов макроса

```
(progn (if(ru-app-load "ru_block_insert_lib")
         (start "000000e7" "ПРИБОР_МЕСТН_КРУГ" 1 1 1 nil)))
```

означает, что блок ПРИБОР_МЕСТН_КРУГ находится в библиотеке 000000e7, вставляется в масштабе 1 (т. е. размер блока в единицах рисунка такой, какой должен быть в миллиметрах на бумаге), и при вставке требуется запрашивать угол поворота. Вызовы макросов

```
(start "ru-lib-valve-ind" "ВЕНТИЛЬ_0" 1 5 5 nil)
(start "ru-lib-valve-ind" "ВЕНТИЛЬ_45" 1 5 5 T)
(start "ru-lib-valve-ind" "ВЕНТИЛЬ_45" 3 5 5 T)
(start "ru-lib-valve-ind" "ВЕНТИЛЬ_45" 5 5 5 T)
(start "ru-lib-valve-ind" "ВЕНТИЛЬ_45" 7 5 5 T)
```

позволяют, используя эту же программу, вставлять изображения вентилей, причем четыре последних варианта вставляют один и тот же блок с разными кодами вставки на разные плоскости аксонометрической схемы. При этом масштаб вставки равен 5 по осям X и Y, первый вариант вставки будет сопровождаться запросом угла поворота.

Конечно, читателям книги эти объяснения кажутся запутанными, но они становятся понятными в практической работе. Как только "подопытная тетка", участвующая в тестировании, пожалуется, что "вентиль слишком большой", мы ей ласково объясняем, где это нужно исправить, и как. Ей достаточно знать, что можно скопировать строку в XML-файле, исправить числа 5 на 2.5, назвать пункт меню "Маленький вентиль" и получить еще один вариант изображения. Возможно, объяснить на примере придется еще раз десять, но остальные 500 вариантов она уже откорректирует сама.

Далее приведем несколько примеров самых ходовых мини-программ.

Вставка блоков различных видов

Варианты функций вставки блоков мы разбирали в *главе 22*. Практически все они "оборачиваются" в программы с простой передачей аргументов в функцию (см. листинг 26.1). Возможны и другие варианты (листинг 26.4), при которых внутри программы какой-то аргумент функции является постоянным.

Листинг 26.4. Файл ru_draw_orient_block.lsp

```
(defun START (block_lib_name block_name X Y is_ask_rotate)
 (ru-app-begin)
  (ru-block-insert-align block_lib_name block_name X Y is_ask_rotate NIL)
 (ru-app-end)(princ))
```

Рисование трасс

Некоторые программы для рисования трасс мы также "проходили":

- `ru_draw_trass_text_ltype` — рисование трасс текстовым типом линии;
- `ru_draw_trass_ltype` — рисование заданным типом линии.

Приведем еще один пример, чтобы электрики не чувствовали себя совсем забытыми (листинг 26.5).

Листинг 26.5. Файл ru_draw_trass_line.lsp

```
(defun START (msg ltype lineweight can_chanche_lw arc_enabled)
  (ru-trass-draw-lw-ltype-from-pt1-arc
    ltype msg arc_enabled can_chanche_lw lineweight 0)
  (princ))
```

Применение этой программы продемонстрировано в листинге 26.6.

Листинг 26.6. Фрагмент файла e_prov.ruxm

```
<?xml version='1.0' encoding='windows-1251'?>
<root name='Проводки электрические' macro=''>
 <item name='Проводники' macro=''>
 <item name='Проводник голый' image='EO\PROVOD\PR_GOL.GIF'
  comment='Рисование трассы специальным типом линии'
  macro='(prong(if(ru-app-load "ru_draw_trass_line")(start "проводника" "WIRE_0"
(ru-lw-current) T  nil)))'/>
 <item name='Проводник 2-линии' image='EO\PROVOD\PR_2.GIF'
  comment='Рисование трассы специальным типом линии'
  macro='(progn(if (ru-app-load "ru_draw_trass_line")(start "проводника"
"WIRE_2_RU" (ru-lw-current) T nil)))'/>
 <item name='Проводник 3-линии' image='EO\PROVOD\PR_3.GIF'
  comment='Рисование трассы специальным типом линии'
  macro='(progn(if (ru-app-load "ru_draw_trass_line")(start "проводника"
"WIRE_3_RU" (ru-lw-current) T nil)))'/>
  </item>
</root>
```

Рисование любых таблиц

Таблицы широко используются во всех разделах проекта. Мы уже приводили примеры нескольких программ для создания таблиц:

- `ru_draw_table_docs` — заполнение ведомости ссылочных и прилагаемых документов;
- `ru_draw_table_user` — создание и разграфка собственных таблиц;
- `ru_pro_table_gen` — генератор таблиц.

В случаях, когда таблица имеет постоянное количество строк, ее можно просто вставить в виде блока (листинг 26.7).

Листинг 26.7. Файл ru_block_insert_table.lsp

```
(defun START (blk)(ru-block-insert-table blk)(princ))
```

Пример элемента XML-меню показан в листинге 26.8.

Листинг 26.8. Фрагмент XML-меню

```
<item name='Основные показатели' image='OV\TABOPOV.GIF'
 comment='Вставляет таблицу в рисунок'
macro='(progn(if(ru-app-load "ru_block_insert_table")(start "tabopov")))'
/>
```

Чаще всего применяется программа, приведенная в листинге 26.9. Мы используем ее для рисования практически всех таблиц с переменным количеством строк и разграфкой.

Листинг 26.9. Файл ru_draw_table.lsp

```
(defun START (form_name is_vertical)
  (ru-app-begin)
  (ru-table-draw-with-ask form_name NIL is_vertical)
  (ru-app-end)
  (princ)
)
```

Пример использования показан в листинге 26.10. В нем имеется один пункт для рисования обычной вертикальной таблицы и два пункта для рисования горизонтальных таблиц — заготовок продольных профилей[1] надземных и подземных газопроводов.

Листинг 26.10. Файл tabl_gs.ruxm

```
<?xml version='1.0' encoding='windows-1251'?>
<root name='Таблицы ГАЗ' macro=''>
 <item name='Осн. показатели ГСВ' image='GAZ\TABGS1.GIF'
  comment='Рисует таблицу заданной длины с автоматической разграфкой'
  macro='(progn(if(ru-app-load "ru_draw_table")(start "tabgs1" T)))'/>
 <item name='Профиль ГСН подземный' image='GAZ\PROGS_P.GIF'
  comment='Рисует заготовку профиля'
  macro='(progn(if(ru-app-load "ru_draw_table")(start "progs_p" nil)))'/>
 <item name='Профиль ГСН надземный' image='GAZ\PROGS_N.GIF'
  comment='Рисует заготовку профиля'
  macro='(progn(if(ru-app-load "ru_draw_table")(start "progs_n" nil)))'/>
</root>
```

[1] Автоматизированного построения профилей в составе нашей системы пока нет, но и простейшие заготовки значительно облегчают работу, особенно когда приходится рисовать километры трасс.

Замечание

В системе AutoCAD 2005 появилось долгожданное средство для работы с таблицами (команда TABLE), позволяющее создавать таблицы и редактировать их содержание. Но так удобно работать с индивидуальными таблицами, а при массовом использовании стандартных таблиц наши программы еще долго будут востребованы. К сожалению, пока нельзя сохранить во внешнем файле определение таблицы AutoCAD 2005 для повторного применения, да и сами таблицы пока примитивные и требуют настройки по месту. Однако объект `Table` в AutoCAD 2005 имеет много свойств и методов, и наверняка мы сможем в будущем разработать программы, позволяющие, используя описание таблиц во внешнем файле, легко создавать и типовые таблицы.

Как научить пользователя добавлять свои команды

Итак, темпы развития нашей системы во многом зависят уже не от программистов, а от продвинутых пользователей. Программистам хватит работы на несколько лет вперед, т. к. на следующем этапе придется переходить к разработке более сложных программ. Массовые операции уже могут обеспечить пользователи, но этому их нужно научить. Практика показывает, что многие программисты этого не умеют делать и не любят. Вскользь мы об этом уже писали. Работа с пользователями является особым искусством, и не все разработчики с ней справляются просто из-за особенностей своего характера. Существуют и чисто технические приемы, позволяющие добиться желаемого результата.

- *Во-первых*, надо для этой достаточно нудной и монотонной работы выбирать не молодых и шустрых ребят, все знающих об AutoCAD, а скромных "теток", склонных к аккуратному выполнению простой работы и теряющихся в ситуациях, когда надо "шевелить мозгами".
- *Во-вторых*, при обучении нужно начинать не с работы в AutoCAD, а с ликвидации пробелов в умении работать с файловым менеджером и текстовым редактором — наверняка этому их никто не удосужился научить. Разумеется, надо обеспечить их русифицированными программами.
- *В-третьих*, не пытайтесь изложить им теоретические основы XML — объясняйте на уровне "скопировать строку, откорректировать то, что в кавычках". Только после получения практических результатов можно будет постепенно объяснить и про элементы, и про теги, и про атрибуты.
- *В-четвертых*, убедитесь, что человек действительно умеет работать в AutoCAD и знает элементарные вещи — использование объектных привязок, ввод данных с клавиатуры и т. п. Проверять точность рисования сотен блоков будет некому и лучше потратить некоторое время на первоначальное обучение.

Наш опыт показывает, что уговорить пользователей включиться в работу и научить их редактировать меню — это не самое трудное. Гораздо сложнее им запомнить, "где что лежит". Каждый конкретный пользователь использует не так уж много команд из нашего меню, из тысяч доступных изображений используются только десятки любимых. Но любимые у каждого разные. Иногда возникают трудности с разыскиванием редко применяемого изображения — точно известно, что оно есть, а в каком меню — забыли.

Напрашивается создание специального меню наподобие **Избранного** в меню программы Internet Explorer, которое каждый пользователь может формировать по вкусу, не занимаясь редактированием текстов меню — все-таки это может не каждый. К сожалению, эту идею мы до сих пор не успели реализовать — просто руки не доходят. Для реализации проще всего в момент работы выбора макроса из XML-меню запоминать его текст в глобальной переменной и, при щелчке по специальной кнопке панели инструментов **Добавить в избранное**, добавлять текст последнего макроса в специальное меню **Избранное**. По такому принципу в системе BestIA мы формировали списки любимых файлов, слоев, калек. Новые возможности открываются в системе AutoCAD 2005 с появлением возможности добавлять в окно Tool Palettes (Инструментальные палитры) собственные команды. Палитры сохраняются в текстовых файлах формата XML, а это означает, что мы сможем создавать их программным путем. Однако эта технология требует дополнительных исследований и применять ее мы будем в следующей версии нашей системы, полностью ориентированной на систему AutoCAD 2005.

ГЛАВА 27

Примеры программ для архитектурно-строительной части

В этой главе мы рассмотрим несколько примеров "рисовальных" программ для строительной части. Программ для строителей разрабатывается очень много, мы разберем только несколько характерных примеров.

Рисование координационных осей

С вычерчивания *координационных осей* здания начинается разработка любых рабочих чертежей. Программы для рисования осей пишут, наверное, все программисты, связанные со строительной тематикой. Мы предложим свой вариант. Программа имеет диалоговое окно, написанное на языке DCL (рис. 27.1).

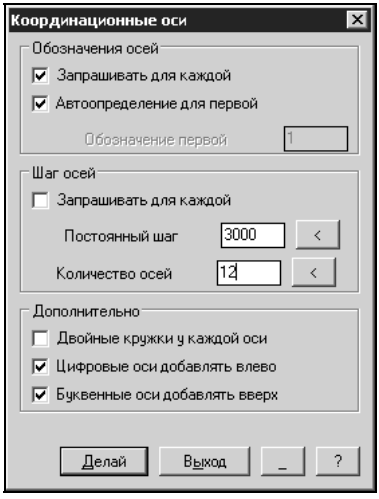


Рис. 27.1. Диалоговое окно программы рисования координационных осей

Напомним, что координационные оси могут быть нарисованы и в виде сетки на плане, и по отдельности на фрагментах и разрезах. Обычно строится прямоугольная

сетка осей, но могут быть и варианты с угловым расположением. Рисование осей выполняется по ГОСТ 21.101—97. Координационные оси наносят на изображения здания, сооружения тонкими штрихпунктирными линиями с длинными штрихами, обозначают арабскими цифрами и прописными буквами русского алфавита (за исключением букв: Ё, З, Й, О, Х, Ц, Ч, Щ, Ъ, Ы, Ь) в кружках диаметром 6—12 мм. Цифрами обозначают координационные оси по стороне здания и сооружения с большим количеством осей. Если для обозначения координационных осей не хватает букв алфавита, последующие оси обозначают двумя буквами (АА, ББ, ВВ). Для обозначения координационных осей блок-секций жилых зданий применяют индекс "с" (1с, 2с, Ас, Бс).

Разумеется, запрограммировать рисование отрезка с кругом и символом внутри очень просто. Сложнее сделать удобную программу. На рисунке диалогового окна видно, что пользователь может задать:

- автоматическое обозначение осей или выбор обозначения по запросу программы;
- постоянный или запрашиваемый шаг осей (возможно указание в рисунке);
- количество рисуемых осей;
- направление прирастания обозначений "буквенных" и "цифровых" осей.

Самым сложным в этой программе является правильное вычисление буквенных обозначений осей, особенно при большом их количестве, а самым нудным — разработка диалогового окна средствами DCL и "издевательство" над стандартным форматированием при вписывании исходного текста в формат книги.

Текст программы приведен в листинге 27.1.

Листинг 27.1. Файл ru_ar_laying_axis.lsp

```
(defun START ( / auto_num  auto_num_first dcl_idx double is_alpha_std_num
 is_ask_all_dist is_digit_std_num l n number1 what_next _toggle_is_alpha_std_num
_toggle_is_digit_std_num _toggle_is_ask_all_dist _toggle_auto_num_first
_toggle_double _toggle_auto _test _draw _draw_axis _auto_num_axis)
;|-------------------------------------------------------------------------
Самая интересная локальная функция автоматического определения обозначения
следующей оси по заданному обозначению предыдущей оси
-------------------------------------------------------------------------|;
  (defun _auto_num_axis (axis_name / base_name flag enabled_chars rus_enabled_chars
lat_enabled_chars axis_section)
;; Список допустимых русских букв
  (setq rus_enabled_chars (list "А" "Б" "В" "Г" "Д" "Е" "Ж" "И" "К" "Л" "М" "Н" "П"
"Р" "С" "Т" "У" "Ф" "Х" "Ц" "Ч" "Ш" "Щ" "Э" "Ю" "Я")
   lat_enabled_chars (list "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N"
"O" "P" "Q" "R" "S" "T" "U" "V" "W" "X" "Y" "Z")
  axis_name
   (if (not axis_name) ""
    (progn
     (setq axis_section (if (= "c" (substr axis_name (strlen axis_name)))
        (progn (setq axis_name
          substr axis_name 1 (1- (strlen axis_name)))) T) nil))
     (strcase axis_name))) base_name "")
```

```
  (if (or (= 'INT (type (read axis_name)))
    (while
       (and (/= axis_name "")
         (setq flag
           (/= 'INT
            (type (read (cond
                        ((or (= "." (substr axis_name 1 1))
                             (= "-" (substr axis_name 1 1))
                        )
                        (setq base_name (strcat base_name
                                          (substr axis_name 1 1))
                              axis_name (substr axis_name 2))
                       )
                        (T axis_name)))))))
       (setq base_name (strcat base_name (substr axis_name 1 1))
            axis_name (substr axis_name 2))
       (if (/= axis_name "") flag nil)
     )
   )
;| Закончилось условие: если нашли в конце строки число, то увеличим его и все!|;
  (setq base_name (strcat base_name (itoa (1+ (read axis_name)))))
;;; А если числа не нашли, то ...
   (progn
        (cond
          ((= 1 (setq flag (strlen base_name)))
           (setq base_name
                (cadr (member base_name
                       (setq enabled_chars
                          (if (member base_name lat_enabled_chars)
                                              lat_enabled_chars
                                              rus_enabled_chars
                          ))));_ end of cadr
                 base_name
                  (if (not base_name) (ru-get-string-cml
                    "Конец списка букв. Введи новое обозначение"
                    (strcat (last rus_enabled_chars) (itoa flag))
                              nil);_ end of ru-get-string-cml
                          base_name
                );_ end of if
          );_ end of setq
         )
          ((< 1 flag)
          ;; Символов - больше одного
           (setq base_name (cadr
                  (member (substr base_name 1 1) enabled_chars)))
                    (if base_name
                      (repeat (1- flag)
                        (setq base_name
                          (strcat base_name (substr base_name 1 1)))
                       );_ end of repeat
```

```
                    (setq base_name (ru-get-string-cml
                        "Конец списка букв. Введи новое обозначение"
                         (strcat (last rus_enabled_chars) (itoa flag))
                            nil);_ end of ru-get-string-cml
            )))
          (T (setq base_name "S1"))
       );_ end of cond
     );_ end of progn
   );_ end of if
    (if axis_section (strcat base_name "c") base_name)
 );_ end of defun
;|-------------------------------------------------------------------
Локальная функция рисования единичной оси
-------------------------------------------------------------------|;
 (defun _draw_axis (start_line_pnt center_circle_pnt axis_txt auto_num
    double / radius)
  (if (not auto_num)
      (setq axis_txt (ru-get-string-cml "Обозначение оси" axis_txt nil))
 );_ end of if
  (setq radius (* (ru-normal-text-height) 2))
  (ru-line-add start_line_pnt
      (polar start_line_pnt
             (angle start_line_pnt center_circle_pnt)
             (- (distance start_line_pnt center_circle_pnt) radius))
      0 "CENTER2");_ end of ru-line-add
  (ru-circle-add center_circle_pnt radius 0 nil)
  (if double (ru-circle-add center_circle_pnt
       (+ radius (ru-conv-millimeter-in-paper-to-unit 1)) 0 nil))
 (ru-text-add axis_txt center_circle_pnt (ru-normal-text-height) 0 "M")
 (_auto_num_axis axis_txt)
);_ end of defun
;|-------------------------------------------------------------------
Локальная функция рисования массива осей
-------------------------------------------------------------------|;
 (defun _draw (txt n dist auto_num auto_num_first double is_ask_all_dist
 is_digit_std_num is_alpha_std_num / ang is_digit_axis msg msg1 msg2 msg3 start_pnt
center_circle_pnt  txt_1)
;|-------------------------------------------------------------------
Аргументы:
Txt- марка 1 оси
n - >= 1  количество осей
dist - шаг осей
auto_num - T|nil - автонумерация без запроса
auto_num_first - T|nil - автонумерация 1-й оси
double -  двойной кружок
is_ask_all_dist - запрос каждого шага
is_digit_std_num - стандартная нумерация цифровых осей
is_alpha_std_num - стандартная нумерация буквенных осей
-------------------------------------------------------------|;
(setq ang 0)
 (if (> n 1)
  (setq msg1 (strcat "Количество осей " (itoa N) ". ") msg2 " первой")
```

```
  (setq msg1 "" msg2 "")
);_ end of if
 (if auto_num_first
      (setq msg3 "")
      (setq msg3 (strcat " '" txt "'"))
);_ end of if
 (setq msg (strcat "\n" msg1 "Начало" msg2 " оси" msg3))
 (if (setq start_pnt (ru-get-point-or-exit msg nil))
   (progn
    (setq center_circle_pnt
                    (ru-get-point-reguired "Центр кружка" start_pnt)
       ang (angle center_circle_pnt start_pnt))
   (if (or
       (<= (* 0.75 pi) (angle start_pnt center_circle_pnt) (* 1.25 pi))
       (<= (* 0.75 pi) (angle center_circle_pnt start_pnt) (* 1.25 pi))
      );_ end of or
       (setq is_digit_axis nil txt_1 "A")
       (setq txt_1 "1" is_digit_axis T)
  );_ end of if
   (if auto_num_first (setq txt txt_1))
   (setq txt
     (_draw_axis start_pnt center_circle_pnt txt auto_num_first double))
;; Вернула обозначение следующей оси
;; Выясняем направление осей
 (if (> n 1)
   (progn
    (if is_digit_axis
       (if is_digit_std_num
          (setq ang (ru-geom-go-right ang))
          (if (ru-dlg-yes-cml "Следующие оси вправо от первой")
              (setq ang (ru-geom-go-right ang))
              (setq ang (ru-geom-go-left ang))));_ end of if
       (if is_alpha_std_num
             (setq ang (ru-geom-go-left ang))
             (if (ru-dlg-yes-cml "Следующие оси влево от первой")
               (setq ang (ru-geom-go-left ang))
               (setq ang (ru-geom-go-right ang)))));_ end of if
            (repeat (- n 1)
             (if is_ask_all_dist
               (setq dist (ru-get-dist
                "Расстояние до следующей оси" dist center_circle_pnt))
             );_ end of if
             (setq start_pnt (polar start_pnt ang dist)
               center_circle_pnt (polar center_circle_pnt ang dist)
               txt
             (_draw_axis start_pnt center_circle_pnt txt auto_num double)
             );_ end of setq
           );_ end of repeat
         );_ end of progn
       );_ end of if
     );_ end of progn
```

```
   );_ end of if
 );_ end of defun
;|-----------------------------------------------------------------------
Далее мелкие неинтересные функции для работы с диалоговым окном
-----------------------------------------------------------------------|
 (defun _test (/ res)
  (setq res T) (mode_tile "button_do" 0) (set_tile "error" "")
  (set_tile "no_auto" (ru-conv-bool-to-str auto_num))
  (set_tile "is_digit_std_num" (ru-conv-bool-to-str is_digit_std_num))
  (set_tile "is_alpha_std_num" (ru-conv-bool-to-str is_alpha_std_num))
  (set_tile "double" (ru-conv-bool-to-str double))
  (set_tile "auto1" (ru-conv-bool-to-str auto_num_first))
  (mode_tile "ed_name" (ru-conv-bool-to-str auto_num_first))
  (set_tile "is_ask_all_dist" (ru-conv-bool-to-str is_ask_all_dist))
  (mode_tile "ed_l" (ru-conv-bool-to-str is_ask_all_dist))
  (mode_tile "get_l" (ru-conv-bool-to-str is_ask_all_dist))
  (setq l       (atof (get_tile "ed_l"))
        n       (atoi (get_tile "ed_n"))
        number1 (get_tile "ed_name")
 );_ end of setq
  (if (<= n 0)
    (progn (mode_tile "button_do" 1)
     (set_tile "error" "Количество осей должно быть не менее 1")
     (setq Res nil)
 ))
  (if (and (<= l 0.0) (> n 1))
    (progn (mode_tile "button_do" 1)
      (set_tile "error" "Неверный шаг нескольких осей")
      (setq Res nil)))
    res
 );_ end of defun

  (defun _toggle_Auto (val) (setq auto_num (= val "0")) (_test))

  (defun _toggle_Double (val) (setq Double (= val "1")) (_test))

  (defun _toggle_auto_num_first (val) (setq auto_num_first (= val "1"))
   (_test))

  (defun _toggle_is_ask_all_dist (val) (setq is_ask_all_dist (= val "1"))
   (_test))

  (defun _toggle_is_digit_std_num (val)
   (setq is_digit_std_num (= val "1")) (_test))

  (defun _toggle_is_alpha_std_num (val)
   (setq is_alpha_std_num (= val "1")) (_test))

;|-----------------------------------------------------------------
Главная функция
-----------------------------------------------------------------|;
 (ru-app-begin)
 (setq L (atof (ru-user-read-last-param "ШагКолонн" "6000.0"))
```

```
  N 1 Number1 "1" auto_num nil auto_num_first T Double nil is_ask_all_dist nil
is_digit_std_num T is_alpha_std_num T
 what_next 5)
(if (> (setq dcl_idx (load_dialog (ru-file-dcl "ru_ar_laying_axis"))) 0)
 (progn
 (while (> what_next 0)
  (if (new_dialog "axis" DCL_IDX)
   (progn
    (set_tile "ed_l" (rtos l 2 0)) (set_tile "ed_n" (itoa N))
    (set_tile "ed_name" Number1) (_test)
    (action_tile "ed_l" "(_Test)") (action_tile "ed_n" "(_Test)")
    (action_tile "ed_name" "(_Test)")
    (action_tile "no_auto" "(_toggle_auto_num $value)")
    (action_tile "double" "(_toggle_Double $value)")
    (action_tile "auto1" "(_toggle_auto_num_first $value)")
    (action_tile "is_ask_all_dist" "(_toggle_is_ask_all_dist $value)")
    (action_tile "is_digit_std_num" "(_toggle_is_digit_std_num $value)")
    (action_tile "is_alpha_std_num" "(_toggle_is_alpha_std_num $value)")
    (action_tile "get_l" "(done_dialog 3)")
    (action_tile "get_n" "(done_dialog 5)")
    (action_tile "button_do" "(done_dialog 4)")
    (action_tile "help" "(ru-help-show \"ru_ar_laying_axis\")")
    (action_tile "cancel" "(done_dialog 0)")
    (setq what_next (start_dialog))
    (cond
      ((= 3 what_next) (setq l (ru-get-dist "Шаг осей" l nil)))
      ((= 5 what_next)
       (setq n (ru-get-int-from-dwg "Укажи место расположения оси" n)))
      ((= 4 what_next)
       (draw Number1 n l auto_num auto_num_first double is_ask_all_dist
         is_digit_std_num is_alpha_std_num)
     )
     ((= 8 what_next) (ru-dcl-hide-dialog))
  );_ end of cond
 );_ end of prong
  (progn
   (setq what_next 0) (ru-msg-alert "Не могу показать диалоговое окно!")
 )
 );_ end of if
);_ end of while
 (ru-user-write-last-param "ШагКолонн" (rtos L 2 2))
 (unload_dialog dcl_idx)
);_ end of progn
 (ru-msg-alert
   (strcat "Не могу загрузить " (ru-file-dcl "ru_ar_laying_axis")))
);_ end of if
(ru-app-end)
(princ)
);_ end of defun
(START)
```

Исходный текст диалогового окна программы приведен в листинге 27.2.

Листинг 27.2. Файл ru_ar_laying_axis.dcl

```
dcl_settings : default_dcl_settings { audit_level = 0; }
@include "ru_cad_lib.dcl"
axis : dialog {label="Координационные оси";
:column {:boxed_column {label = "Обозначения осей";
: toggle
 {label = "Запрашивать для каждой"; key = "no_auto"; value = "0";}
: toggle
{label = "Автоопределение для первой"; key = "auto1"; value = "0";}
 : edit_box
{fixed_width = true; width = 10; alignment = right;
label = "Обозначение первой";key = "ed_name";edit_width = 6;}}
:boxed_column
{label = "Шаг осей";: toggle {label = "Запрашивать для каждой";
 key = "is_ask_all_dist";value = "0";}
:row {fixed_width = true; width = 30;children_alignment = right;
:spacer_1 {width=4;}: edit_box {label = "Постоянный шаг";key = "ed_l";
 edit_limit = 6; edit_width = 6;}
: button {label = "<";key = "get_l";fixed_width = true; width = 1;}}
:row {fixed_width = true; width = 30; children_alignment = right;
:spacer_1 {width=3;}: edit_box {label = "Количество осей";key = "ed_n";
edit_limit = 3; edit_width = 6;}
: button {label = "<";key = "get_n";fixed_width = true; width = 1;}}}
:boxed_column {label = "Дополнительно";
: toggle {label = "Двойные кружки у каждой оси";}
: toggle {label = "Цифровые оси добавлять влево";        value = "1";}
: toggle {label = "Буквенные оси добавлять вверх";
key = "is_alpha_std_num";value = "1";}}}
 errtile; do_exit_hide_help;}
```

В результате работы программы может быть нарисована сетка осей, подобная показанной на рис. 27.2.

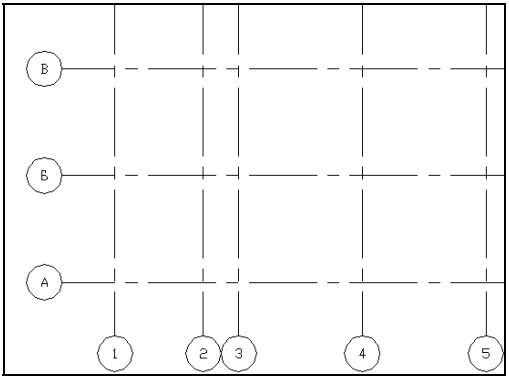


Рис. 27.2. Результат работы программы рисования координационных осей здания

Мы не предусматриваем автоматической простановки размеров между осями, т. к. это удобнее делать стандартными средствами системы AutoCAD "по вкусу" на требуемых слоях. Эта программа назначена в качестве *метода слоя (см. главу 18)* в классификаторе слоев и может быть выполнена при вызове команды "Создать объект методом слоя", если установлен соответствующий слой.

Рисование стен и перегородок

После рисования осей было бы естественным заняться разработкой программы рисования *стен*. Однако самый основной рисунок для строительной части — рисование стен — мы автоматизировать не будем. Что есть стена или перегородка в двухмерном черчении — просто две параллельных линии. Параллельны они сами себе и какой-то базовой линии, обычно координационной оси или другой стене. Мы делали много вариантов рисования двойных линий, и до того, как в системе AutoCAD R11 впервые появилась команда DLINE, и после появления мультилиний. В доказательство приводим рис. 27.3, на котором показано диалоговое окно одной из программ. Судя по его виду, программа может рисовать любые прямолинейные стены (может делать и врезки в случае примыкания к ранее нарисованным). Конечно, это не такие стены, которые создаются в ADT[1] или хотя бы в системе АРКО — это обычные полилинии.

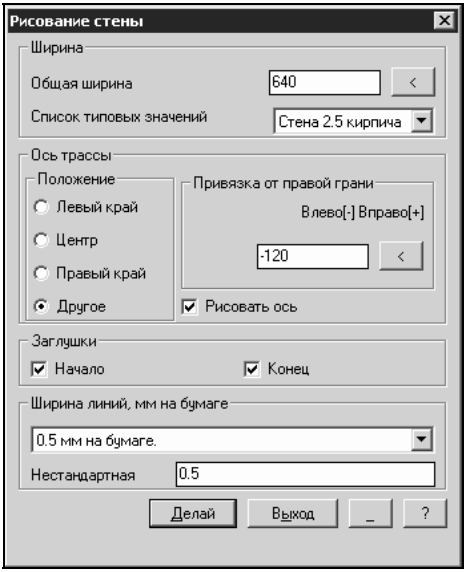


Рис. 27.3. Диалоговое окно рисования стены

Но чем больше мы разрабатывали вариантов этой программы, тем меньше видели необходимость в ней. То ли строители и архитекторы нам попадались какие-то "не-

[1] Autodesk Architectural Desktop.

правильные", то ли еще что, но они предпочитают работать просто и эффективно — создавать контуры стен просто командой OFFSET (ПОДОБИЕ) от существующих объектов с последующими сопряжениями, удлинениями и обрезками.

Вместо этой большой программы, которая займет слишком много места, мы разработаем маленькую программу для дорисовки пилястр и ниш к стенам, нарисованными отрезками или полилиниями. Эта программа может послужить примером изменения технологии программирования. Когда-то мы разрабатывали ее с диалоговым окном (рис. 27.4), в котором можно было задать два размера и выбрать ширину линии.

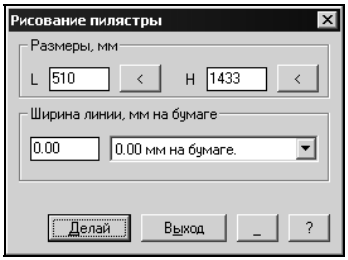


Рис. 27.4. Диалоговое окно программы рисования пилястр

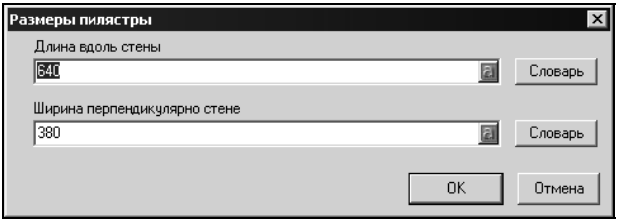


Рис. 27.5. Ввод размеров пилястр

Программ, в которых нужно вводить каких-то два размера, требуется довольно много. При использовании DCL можно программно изменять значения меток диалога (L, H заменить на "Длина" и "Ширина"), но не очень удобно — приходится писать дополнительный код. Когда нам это надоело, мы вернулись к любимой командной строке, и это оказалось не худшим решением. Кроме того, мы ведь разработали универсальный диалог ввода двух строк *(см. главу 15)*, который можем использовать и для ввода чисел. Вызвав функцию ввода двух строк

```
(ru-dlg-get-two-string "Размеры пилястры" "Длина вдоль стены" "640" T
    "Ширина перпендикулярно стене" "380" T T "options\\wall_w.xml"
    "options\\wall_w.xml")
```

мы выведем на экран диалоговое окно (рис. 27.5), в котором можем ввести размеры вручную, указать их в рисунке или выбрать из XML-меню типовых размеров стен (рис. 27.6).

При выборе из справочника толщины кирпичной стены 2.5 кирпича мы получим число 640, а диалоговое окно вернет список ("640" "380"). Конечно, пользователь может ввести и неправильное число, и вообще любую строку, но контроль за вводом нам придется выполнить в основной программе, а еще лучше, на базе функции `ru-dlg-get-two-string` разработать функцию `ru-dlg-get-two-number` (листинг 27.3), осуществляющую контроль ввода чисел[1].

[1] А еще мы можем создать аналогичный COM-сервер для ввода чисел, не позволяющий щелкнуть по кнопке **OK**, пока не будут введены допустимые числа.

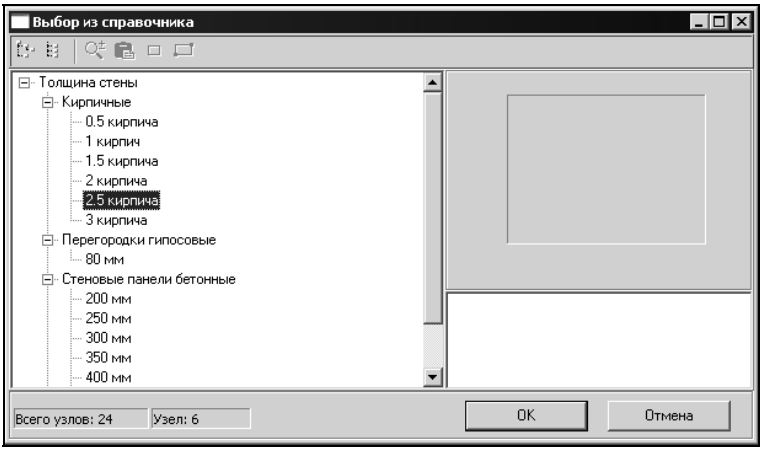


Рис. 27.6. Выбор толщины стены из справочника

Листинг 27.3. Функция `ru-dlg-get-two-number`

```
(defun ru-dlg-get-two-number (caption label1 default1 min1 max1 label2
default2 min2 max2 xml_file_name1 xml_file_name2 / end err_str lst num1 num2 result
str1 str2)
;|-------------------------------------------------------------------------
Ввод двух чисел
-------------------------------------------------------------------------|;
 (while (not end)
  (if (setq lst
    (ru-dlg-get-two-string caption label1 (vl-princ-to-string default1)
     T label2 (vl-princ-to-string default2) T T xml_file_name1
      xml_file_name2)
   );_ end of setq
   (progn
     (setq err_str "" str1 (car lst) str2 (cadr lst) num1 (atof str1)
           num2    (atof str2))
     (if (< num1 min1) (setq err_str (strcat err_str "\n" label1 "=" str1
         " меньше допустимого значения " (vl-princ-to-string min1))))
     (if (> num1 max1) (setq err_str (strcat err_str "\n" label1 "=" str1
         " больше допустимого значения " (vl-princ-to-string max1))))
     (if (< num2 min2) (setq err_str (strcat err_str "\n" label2 "=" str2
         " меньше допустимого значения " (vl-princ-to-string min2))))
     (if (> num2 max2) (setq err_str (strcat err_str "\n" label2 "=" str2
         " больше допустимого значения " (vl-princ-to-string max2))))
     (if (/= err_str "")
       (ru-msg-alert (strcat caption "\nОбнаружены ошибки: " err_str
                     "\nПовторите ввод!"))
       (setq end t result (list num1 num2))
     );_ end of if
  );_ end of progn
```

```
   (setq end T)
 );_ end of if
);_ end of while
 result
)
```

Теперь разработаем основную программу (листинг 27.4), которая всего-навсего рисует П-образную полилинию, но использует несколько очень полезных функций. Главной функции `start` передается два аргумента — название рисуемого объекта и признак необходимости разрыва основной линии, например: `(start "пилястры" T)` или `(start "ниши" nil)`.

Листинг 27.4. Файл ru_ar_union_pilaster_in_line.lsp

```
(defun START (msg is_break /   ang end_point ent ent_point h l lst_dim lst_pnt
pick_point)
  (ru-app-begin)
;; Восстанавливаем предыдущие параметры
  (setq l (ru-conv-millimeter-to-unit (atof
         (ru-user-read-last-param (strcat "Размер" msg "L") "510.0")))
       h (ru-conv-millimeter-to-unit (atof
         (ru-user-read-last-param (strcat "Размер" msg "H") "510.0")))
 );_ end of setq
  (while
   (setq lst_dim (ru-dlg-get-two-number (strcat "Размеры " msg)
       "Длина вдоль стены" l 120 3000 "Ширина перпендикулярно стене" h
       50 6000 "options\\wall_w.xml" "options\\wall_w.xml")
        );_ end of setq
    (setq l (car lst_dim) h (cadr lst_dim))
;; Запрашиваем точку врезки объекта
   (while (setq ent_point (ru-get-point-on-ent
                        (strcat "Начало " msg " на грани стены")))
    (setq ent (car ent_point) pick_point  (cadr ent_point))
;|-------------------------------------------------------------------------
Разыскиваем точки начала и конца примитива ent, указанного в точке.
Получаем список из имени этого же примитива и точек начала и конца. Для полилинии
это начало и конец указанного сегмента
-------------------------------------------------------------------------|;
    (if (setq lst_pnt (ru-geom-list-ent-point ent pick_point))
      (progn
        (setq ang (angle pick_point (nth 2 lst_pnt)))
;|-------------------------------------------------------------------------
Запрашиваем ориентацию объекта относительно указанной точки - достаточно указать
любую точку в нужном направлении. Целиться никуда не надо.
-------------------------------------------------------------------------|;
        (if (ru-get-is-point-right-by-axis
             (strcat " " msg " размером " (rtos l 2 2) " вдоль стены ")
              pick_point (polar pick_point (ru-geom-go-right ang) h)
           );_ end of ru-get-is-point-right-by-axis
          (setq ang (angle pick_point (nth 1 lst_pnt)))
       )
```

```
        (setq  end_point (polar pick_point ang l))
        (if is_break
          (progn
;|-------------------------------------------------------------------------
При необходимости вырезаем кусочек рассчитанной длины
-------------------------------------------------------------------------|;
          (ru-var-clear-osnap)
          (command "_.BREAK" (list ent pick_point) "_F" pick_point
                                                        end_point)
          (ru-var-restore-osnap)
        );_ end of progn
     );_ end of if
;|-------------------------------------------------------------------------
Запрашиваем ориентацию поперек основной линии
-------------------------------------------------------------------------|;
    (if (ru-get-is-point-right-by-axis (strcat " " msg " размером "
          (rtos h 2 2) " поперек стены ") pick_point end_point)
        (setq ang (ru-geom-go-right ang))
        (setq ang (ru-geom-go-left ang)));_ end of if
;|-------------------------------------------------------------------------
Рисуем полилинию
-------------------------------------------------------------------------|;
    (ru-pline-add (list pick_point (polar pick_point ang h)
            (polar end_point ang h) end_point) nil 0 0 nil)
;|-------------------------------------------------------------------------
Устанавливаем для полилинии такие же свойства, как и для основной линии
-------------------------------------------------------------------------|;
   (ru-obj-copy-prop "LINEWEIGHT"
       (vlax-ename->vla-object ent) (vlax-ename->vla-object (entlast)))
   (ru-obj-copy-prop "CONSTANTWIDTH"
       (vlax-ename->vla-object ent) (vlax-ename->vla-object (entlast)))
 );_ end of progn
);_ end of if
);_ end of while
);_ end of if
  (ru-user-write-last-param (strcat "Размер" msg "L") (rtos l 2 2))
  (ru-user-write-last-param (strcat "Размер" msg "H") (rtos h 2 2))
  (ru-app-end)
  (princ)
)
```

При копировании свойств от основной линии (отрезка или компактной полилинии) к вновь нарисованному объекту не возникнет сбойной ситуации, даже если мы попытаемся присвоить отрезку свойство `ConstantWidth`, которого у отрезка не существует. Объясняется это тем, что в функции `ru-obj-copy-prop` предусмотрена наша традиционная ловушка ошибок `ru-error-catch`.

В результате работы программы могут быть нарисованы пилястры и ниши (рис. 27.7).

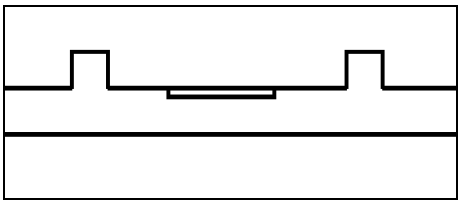


Рис. 27.7. Результаты рисования пилястр и ниши

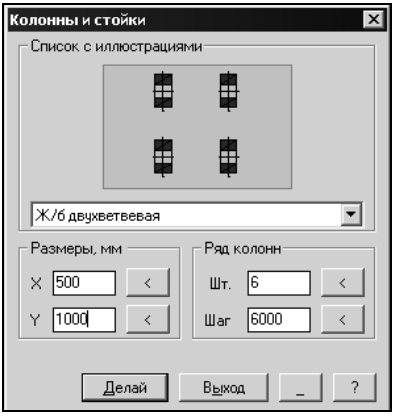


Рис. 27.8. Диалоговое окно программы рисования колонн

Рисование колонн и опор

Еще одним примером упрощения технологии программирования является программа рисования на плане колонн различных конструкций и типоразмеров. Прежде она имела диалоговое окно (рис. 27.8), над которым пришлось немало потрудиться. В диалоговом окне можно было выбрать конструкцию колонны из выпадающего списка, каждая конструкция была проиллюстрирована слайдом (который надо было создать и поместить в библиотеку), имелась возможность ввести или указать размеры, количество и шаг колонн. Владея рассмотренными в книге технологиями, мы можем легко сформировать иллюстрированное XML-меню для любых конструкций и типоразмеров. После выбора типоразмера остается указать шаг и количество изображаемых колонн, а это легко сделать в командной строке.

Программа могла бы вообще только производить множественную вставку блока, тем более что точки вставки в виде пересечений осей имеются, но мы все-таки предусмотрим возможность рисования рядов колонн с заданным шагом, имея в виду, что эта программа нам понадобится не только для колонн, но и для многих иных изображений.

Сначала решим, как мы будем рисовать любую отдельную колонну. Разумеется, лучше всего использовать блоки. Самым простым решением является использование единичных блоков (рис. 27.9) — в этом случае, задавая масштабы вставки по осям X и Y, равные реальным размерам колонн, можно один единичный блок использовать для всех типоразмеров. Обычно это приемлемо, но если важно выдержать и внутриблочные размеры, то могут понадобиться блоки реальных изображений. Например, при использовании единичного блока двухветвевой колонны не совсем верно будет отображаться толщина ветвей, а это может оказаться препятствием при разработке смежных разделов. Очевидно, надо иметь возможность использовать любые варианты рисования, следовательно, макрос XML-меню должен сообщить программе и то, каким методом следует рисовать колонны.

"Продвинутые" пользователи, зная допустимые методы рисования, смогут сами создавать необходимые им блоки и расширять меню. В принципе, колонна может ри-

соваться и не блоком, а какой-нибудь "хитрой" функцией — лишь бы в момент рисования эта функция была определена. Колонны лучше всего рисовать рядами, задавая тип и количество колонн, расстояние между колоннами и направление ряда. Для этого пригодится функция, приведенная в листинге 27.5.

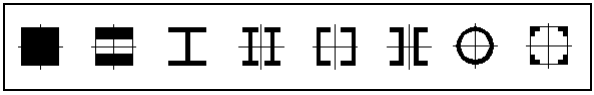


Рис. 27.9. Единичные блоки схематичных изображений колонн

Листинг 27.5. Функция `ru-block-insert-dist-count`

```
(defun ru-block-insert-dist-count
       (block_name x_scale y_scale z_scale first_point ang dist count)
  (repeat count
    (ru-block-insert-obj block_name first_point x_scale y_scale 1 ang)
    (setq first_point (polar first_point ang dist))
 )
  first_point
)
```

Функция, осуществляющая запрос исходных данных для рисования ряда, приведена в листинге 27.6.

Листинг 27.6. Функция `ru-trass-draw-block-count`

```
(defun ru-trass-draw-block-count (block_name x_scale y_scale dist count
                                  / first_point)
  (while (setq first_point (_ru-get-with-default (strcat "Dist="
       (rtos dist) ", Count=" (itoa count) ". Точка начала ряда")
                  "Выход" 'getpoint nil "Dist Count" nil))
    (if (ru-is-point first_point)
      (setq first_point (trans first_point 1 0)))
    (cond
      ((= first_point "Dist")
       (setq dist (ru-get-dist "Пролет" dist nil)))
      ((= first_point "Count")
       (setq count (ru-get-int-or-pick "Количество" count)))
      (T (setq first_point
         (ru-block-insert-dist-count block_name x_scale y_scale 1
           first_point
          (angle first_point
            (ru-get-point-reguired "Направление ряда" first_point))
              dist count)
      )
     )
   )
 )
  (list dist count)
)
```

Основная программа, вызов которой осуществляется из XML-меню, приведена в листинге 27.7.

Листинг 27.7. Файл ru_ar_column_row.lsp

```
(defun START (block_lib block_name x_size_mm y_size_mm / result)
 (ru-app-begin)
 (if (ru-block-lib-insert block_lib block_name)
  (progn
   (setq result (ru-trass-draw-block-count block_name
         (ru-conv-millimeter-to-unit x_size_mm)
         (ru-conv-millimeter-to-unit y_size_mm)
         (ru-conv-millimeter-to-unit
         (atof (ru-user-read-last-param
          (strcat "ColumnDistForBlock" block_name)
                           "6000")))
         (atoi (ru-user-read-last-param
          (strcat "ColumnCountForBlock" block_name) "1"))))
  (ru-user-write-last-param (strcat "ColumnDistForBlock" block_name)
         (rtos (car result)))
  (ru-user-write-last-param (strcat "ColumnCountForBlock" block_name)
         (rtos (cadr result)))
    );_ end of progn
 );_ end of if
  (ru-app-end)
  (princ)
)
```

В результате может быть нарисован ряд колонн, подобный показанному на рис. 27.10.

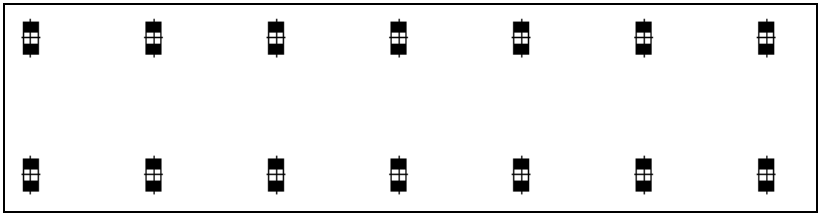


Рис. 27.10. Нарисованные ряды колонн

Пример с колоннами показывает, как можно избежать разработки диалоговых окон и уходить от связанных с этим проблем. Полученная программа может использоваться не только для рисования колонн зданий, но и для любых других объектов, которые можно изобразить расстановкой блоков на заданном расстоянии. Все дальнейшее "программирование" сводится к подготовке необходимых блоков и редактированию XML-меню, а это уже работа, за которую охотно берутся обычные пользователи.

Отверстия в стенах и перекрытиях

Наличие планов с отверстиями для прокладки коммуникаций является одним из признаков качественных рабочих чертежей — независимо от того, предусматривается ли пробивка отверстий или они должны быть оставлены при сооружении стен и перегородок. Отверстия всегда являлись предметом спора как между конструкторами-строителями и смежниками, так и между генподрядчиками и субподрядчиками. В "советские" времена отсутствие отверстий или несоответствие их размеров СНиП по производству работ могло послужить причиной для отказа субподрядчика от приемки объекта под монтаж — напомним, что тогда очень часто все искали не работу, а причины для уклонения от нее.

Раздоры внутри проектных организаций происходят до сих пор. Конструкторам-строителям не хочется делать лишних проемов с перемычками или монолитных участков в перекрытиях, а смежники должны добиться, чтобы отверстия соответствовали нормам. Размеры отверстий по СНиП предусматривают комфортные условия для монтажа коммуникаций, например, размеры отверстий для воздуховодов должны быть на 150 мм больше диаметра или размера стороны воздуховода, следовательно, даже для воздуховода с минимальным диаметром 100 мм уже нельзя воспользоваться пробивкой отверстия в пустотном перекрытии. В результате часто и строители, и смежники делают вид, что проблемы с отверстиями не существует и оставляют ее для решения по принципу "пробивка по месту, диаметр по соображению".

Мы все-таки будем рассматривать вариант нормальной работы — подготовку смежниками задания на отверстия и выполнение строителями планов отверстий. При использовании САПР эти два этапа можно легко совместить. Смежники должны аккуратно нарисовать на специальных слоях планы отверстий, а строители должны с минимальной доработкой использовать эти слои в своих разделах. После сведения в один файл планов отверстий разных марок необходимо составить ведомость отверстий.

Разработаем программу для рисования отверстий для коммуникаций. Здесь нам не обойтись без диалогового окна (рис. 27.11).

Мы видим, что программа позволяет:

- задать размеры отверстия путем ввода в поле редактирования, измерением в рисунке (кнопки <), указанием на отверстие-аналог (кнопка **Аналог**) или выбором из справочника (кнопка **СНиП**);

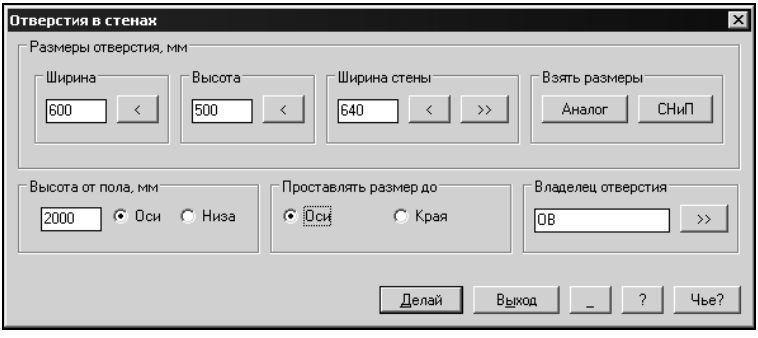


Рис. 27.11. Диалоговое окно программы рисования отверстий в стенах

- задать толщину стены прямым вводом, измерением или выбором из справочника;
- задать отметку оси или низа отверстия от пола этажа;
- указать, до оси или края отверстия выполняется привязка;
- задать марку раздела-владельца отверстия или выбрать ее из справочника.

Во время рисования отверстия (кнопка **Делай**) программа выдаст запросы:

```
Ось отверстия <Выход>:
```

Пользователь указывает начало отверстия в месте входа трубопровода в стену.

```
Направление трассы:
```

Пользователь указывает точку в нужном направлении. Программа рисует зачерненное обозначение отверстия в виде блока, присваивает ему требуемые значения атрибутов, прицепляет к концу отверстия первую точку выносной линии маркировки и запрашивает точки выноски.

```
Вторая точка выносной линии номера отверстия:
Следующая точка  выносной линии номера отверстия  <Выход>:
```

После рисования выноски на ее конце подписывается обозначение отверстия и запрашивается точка привязки оси или края отверстия.

```
Начало выносной размерной линии от конструкции:
```

После указания точки привязки выводится запрос

```
Положение размерной линии:
```

И программа дожидается, пока пользователь не передвинет размерную линию в нужное место (попытки нажатия клавиши <Esc> пресекаются).

В изобразительном смысле программа проста — рисуются залитые прямоугольники, выноски обозначений и размерные привязки, но для последующего составления ведомости отверстий приходится использовать ряд специальных приемов, которые могут послужить прототипами при разработке других программ.

Рисование отверстий в стенах

Ознакомимся с исходным текстом программы (листинг 27.8). В ней используется много мелких локальных функций и несколько специализированных глобальных функций, которые пригодятся для программы рисования отверстий в перекрытиях.

Листинг 27.8. Функция `ru-holes-wall`

```
(defun ru-holes-wall (/*is_axes_level *bottom_level *is_dim_axes *hole_height
*hole_owner *hole_width *level_from_floor *number *max_number *wall_width dcl_idx
dlg_file dlg_name what_next _test _pick_dim _pick_level _change_level _select_dim
_select_owner _select_wall_width _set_tiles _save _restore _hole_info _leader
_analog _draw_hole _draw_dim_rotated)
;;;-------------------------------------------------------------------------
;;; Основная функция рисования отверстий в стенах
;;;-------------------------------------------------------------------------
;;;-------------------------------------------------------------------------
;;; Локальная функция рисования размера
;;;-------------------------------------------------------------------------
(defun _draw_dim_rotated (pnt1 pnt2 ang)
```

```
 (while (not (ru-error-catch (function (lambda ()
  (princ "\nПоложение размерной линии: ")
  (vl-cmdf "_.DIMLINEAR" pnt1 pnt2 "_Rotated" (ru-conv-rad-to-deg ang)
    pause))) nil))))
;;;-------------------------------------------------------------------------
;;; Локальная функция вычисления номера типоразмера отверстия
;;;-------------------------------------------------------------------------
(defun _eval_number (/ number)
 (if (setq number (ru-holes-wall-find-hole-number *hole_width
 *hole_height *wall_width *bottom_level *hole_dic_list*))
 (setq *number number)
 (progn
 (setq *number (1+ *max_number) *max_number *number)
;;; При добавлении нового отверстия список пополняется
 (setq *hole_dic_list* (cons (ru-holes-wall-hole-dic-dot-pair
       *number *hole_width *hole_height *wall_width *bottom_level)
 *hole_dic_list*)))))
;;;-------------------------------------------------------------------------
;;; Локальная функция рисования отверстия
;;;-------------------------------------------------------------------------
(defun _draw_hole (/ start_pnt end_pnt left_pnt om right_pnt sa start_dim_pnt ang
old_lay)
 (setq old_lay (getvar "CLAYER"))
;; Установка слоя владельца отверстия
 (ru-layer-current (strcat "RU_WALL_HOLE_MARK_" *hole_owner))
 (while (setq start_pnt (ru-get-point-or-exit "Ось отверстия" nil))
   (setq end_pnt (ru-get-point-reguired "Направление трассы" start_pnt)
         ang (angle start_pnt end_pnt)
         end_pnt (polar start_pnt ang
            (ru-conv-millimeter-to-unit *wall_width)))
;; Вычисление номера типоразмера
 (_eval_number)
;; Вставка блока отверстия
 (ru-block-insert-obj "RU_WALL_HOLE_MARK" start_pnt
   (ru-conv-millimeter-to-unit *wall_width)
   (ru-conv-millimeter-to-unit *hole_width) 1 (angle start_pnt end_pnt))
;; Автозаполнение атрибутов
 (ru-block-change-attributes  (entlast) (list
  (cons "OWNER" *hole_owner)
  (cons "HEIGHT"
   (vl-princ-to-string (ru-conv-unit-to-millimeter *hole_height)))
  (cons "WIDTH"
   (vl-princ-to-string (ru-conv-unit-to-millimeter *hole_width)))
  (cons "BOTTOM_LEVEL"
   (vl-princ-to-string (ru-conv-unit-to-millimeter *bottom_level)))
  (cons "LENGTH"
   (vl-princ-to-string (ru-conv-unit-to-millimeter *wall_width)))
))
;; Построение выноски с маркой отверстия
 (_leader end_pnt (strcat *hole_owner "-" (itoa *number)))
```

```
;;Образмеривание
 (setq start_dim_pnt (ru-get-point-reguired
   "Начало выносной размерной линии от конструкции" start_pnt)
   sa (getvar "SNAPANG") om (getvar "ORTHOMODE"))
 (setvar "SNAPANG" ang) (setvar "ORTHOMODE" 1)
 (if *is_dim_axes
;;размер до оси
 (_draw_dim_rotated (trans end_pnt 0 1) (trans start_dim_pnt 0 1)
  (ru-geom-go-right ang))
;; Если размер до края
;; выясняем, до какого края
 (if (<= (distance (setq left_pnt (polar end_pnt (ru-geom-go-left ang)
    (/ *hole_width 2))) start_dim_pnt);_ end of distance
     (distance (setq right_pnt (polar end_pnt (ru-geom-go-right ang)
    (/ *hole_width 2))) start_dim_pnt))
  (_draw_dim_rotated (trans left_pnt 0 1) (trans start_dim_pnt 0 1)
   (ru-geom-go-left ang))
  (_draw_dim_rotated (trans right_pnt 0 1) (trans start_dim_pnt 0 1)
   (ru-geom-go-right ang))
))
 (setvar "SNAPANG" sa) (setvar "ORTHOMODE" om))
;; Восстановление слоя
 (ru-layer-current old_lay)
)
;;;----------------------------------------------------------------------
;;; Локальная функция получения параметров отверстия по аналогу
;;;----------------------------------------------------------------------
(defun _analog (/ ent _bottom_level _hole_height _hole_owner _hole_width _number
_wall_width)
 (if (setq ent (ru-get-entsel-by-type "Укажите отверстие" "Это не БЛОК"
       (list "INSERT") t))
 (progn
 (setq ent (car ent))
 (if (and (setq_hole_width (ru-block-attrib-by-name ent "WIDTH")
               _hole_height (ru-block-attrib-by-name ent "HEIGHT")
               _wall_width (ru-block-attrib-by-name ent "LENGTH")
               _bottom_level (ru-block-attrib-by-name ent "BOTTOM_LEVEL")
               _hole_owner (ru-block-attrib-by-name ent "OWNER")))
  (progn
   (setq *hole_width (atof _hole_width) *hole_height (atof _hole_height)
   *wall_width (atof _wall_width) *bottom_level (atof _bottom_level))
;; сменить владельца
   (_change_owner _hole_owner)
   (_eval_number)
   (ru-msg-info  (strcat "ВЗЯТО ЗА АНАЛОГ ОТВЕРСТИЕ:"
    "\n\nМарка раздела " _hole_owner ";\nДлина = " _hole_width
    " мм;\nВысота = " _hole_height " мм;\nТолщина стены = " _wall_width
    " мм;\nНиз отверстия от пола этажа = " _bottom_level " мм "))
 )
 (ru-msg-alert "Это НЕ отверстие в СТЕНЕ!"))))
);_ end of defun
```

```
;;;-------------------------------------------------------------------------
;;; Локальная функция рисования выноски
;;;-------------------------------------------------------------------------
 (defun _leader (start_pnt txt / list_point)
  (if (setq list_point (_ru-trass-draw-lw
      (strcat " выносной линии номера отверстия " txt) start_pnt nil nil
       nil nil 0 0 nil))
   (ru-text-add (strcat "%%u" txt)(trans
     (ru-text-end-leader-point (angle (cadr list_point) (car list_point))
     (car list_point)) 1 0)
     (ru-normal-text-height) 0
     (ru-text-end-leader-align
       (angle (cadr list_point) (car list_point))))
) (princ))
;;;-------------------------------------------------------------------------
;;; Локальная функция справки по отверстию
;;;-------------------------------------------------------------------------
 (defun _hole_info (/ ent _bottom_level _hole_height _hole_owner _hole_width
_wall_width)
 (if (setq ent (ru-get-entsel-by-type "Укажите отверстие" "Это не БЛОК"
 (list "INSERT") T))
 (progn
 (setq ent (car ent))
 (if (and (setq _hole_width (ru-block-attrib-by-name ent "WIDTH")
               _hole_height (ru-block-attrib-by-name ent "HEIGHT")
                _wall_width (ru-block-attrib-by-name ent "LENGTH")
               _bottom_level (ru-block-attrib-by-name ent "BOTTOM_LEVEL")
               _hole_owner (ru-block-attrib-by-name ent "OWNER")))
   (ru-msg-info (strcat "ДАННЫЕ ОТВЕРСТИЯ:"
    "\n\nМарка раздела " _hole_owner ";\nДлина = " _hole_width
    " мм;\nВысота = " _hole_height " мм;\nТолщина стены = " _wall_width
    " мм;\nНиз отверстия от пола этажа = " _bottom_level " мм "))
   (ru-msg-alert "Это НЕ отверстие в СТЕНЕ!")))))
;;;-------------------------------------------------------------------------
;;; Локальная функция восстановления параметров отверстия
;;;-------------------------------------------------------------------------
(defun _restore (/ ini_file)
 (setq ini_file (ru-file-dwg-ini)
  *wall_width (atof (ru-ini-read ini_file "Setup" "wall_width" "640.0"))
  *hole_owner (ru-ini-read ini_file "wall_holes" "hole_owner" "ОВ")
  *hole_dic_list* (ru-holes-wall-read-dic *hole_owner)
  *max_number (atoi (ru-ini-read ini_file "wall_holes"
    (strcat *hole_owner "-max_number") "0"))
  *hole_width (ru-conv-millimeter-to-unit
   (atof (ru-ini-read ini_file "wall_holes" "hole_width" "1000.0")))
  *hole_height (ru-conv-millimeter-to-unit
   (atof (ru-ini-read ini_file "wall_holes" "hole_height" "1000.0")))
  *is_axes_level (ru-conv-str-to-bool
   (ru-ini-read ini_file "wall_holes" "is_axes_level" "1"))
  *is_dim_axes (ru-conv-str-to-bool
    (ru-ini-read ini_file "wall_holes" "is_dim_axes" "1"))
```

```
  *level_from_floor (ru-conv-millimeter-to-unit
  (atof (ru-ini-read ini_file "wall_holes" "level_from_floor" "1000.0"))
))
 (_change_level)
);_ end of defun
;;;-------------------------------------------------------------------------
;;; Локальная функция сохранения параметров отверстия
;;;-------------------------------------------------------------------------
(defun _save (/ ini_file)
 (ru-holes-wall-write-dic *hole_owner *hole_dic_list*)
 (setq ini_file (ru-file-dwg-ini))
 (ru-ini-write ini_file "Setup" "wall_width" (rtos *wall_width 2 2))
 (ru-ini-write ini_file "wall_holes" "hole_owner" *hole_owner)
 (ru-ini-write ini_file "wall_holes" (strcat *hole_owner "-max_number")
  (itoa *max_number))
 (ru-ini-write ini_file "wall_holes" "hole_width"
  (rtos (ru-conv-unit-to-millimeter *hole_width) 2 2))
 (ru-ini-write ini_file "wall_holes" "hole_height"
  (rtos (ru-conv-unit-to-millimeter *hole_height) 2 2))
 (ru-ini-write ini_file "wall_holes" "is_axes_level"
  (ru-conv-bool-to-str *is_axes_level))
 (ru-ini-write ini_file "wall_holes" "is_dim_axes"
  (ru-conv-bool-to-str *is_dim_axes))
 (ru-ini-write  ini_file "wall_holes" "level_from_floor"
  (rtos (ru-conv-unit-to-millimeter *level_from_floor) 2 3))
 (princ)
)
;;;-------------------------------------------------------------------------
;;; Локальная функция заполнения полей диалогового окна
;;;-------------------------------------------------------------------------
(defun _set_tiles ()
 (set_tile "ed_hole_owner" *hole_owner)
 (set_tile "ed_hole_height"
  (rtos (ru-conv-unit-to-millimeter *hole_height) 2 0))
 (set_tile "ed_hole_width"
  (rtos (ru-conv-unit-to-millimeter *hole_width) 2 0))
 (set_tile "ed_wall_width"
  (rtos (ru-conv-unit-to-millimeter *wall_width) 2 0))
 (set_tile "ed_level"
  (rtos (ru-conv-unit-to-millimeter *level_from_floor) 2 0))
 (set_tile "axes_level" (ru-conv-bool-to-str *is_axes_level))
 (set_tile "bottom_level" (ru-conv-bool-to-str (not *is_axes_level)))
 (set_tile "dim_axes" (ru-conv-bool-to-str *is_dim_axes))
 (set_tile "dim_border" (ru-conv-bool-to-str (not *is_dim_axes)))
 (_test)
 (princ)
)
;;;-------------------------------------------------------------------------
;;; Локальная функция реакции на изменение владельца отверстия
;;;-------------------------------------------------------------------------
(defun _change_owner (new_owner)
 (ru-ini-write (ru-file-dwg-ini) "wall_holes"
  (strcat *hole_owner "-max_number") (itoa *max_number))
```

```
;; Сохраняем данные отверстий прежнего владельца
 (ru-holes-wall-write-dic *hole_owner *hole_dic_list*)
 (setq *hole_owner new_owner
;; Читаем данные нового владельца
 *hole_dic_list* (ru-holes-wall-read-dic *hole_owner)
 *max_number (atoi (ru-ini-read (ru-file-dwg-ini) "wall_holes"
 (strcat *hole_owner "-max_number") "0")))
)
;;;-------------------------------------------------------------------------
;;; Локальная функция выбора владельца из словаря
;;;-------------------------------------------------------------------------
(defun _select_owner (/ data)
 (if (setq data (ru-xml-get-sdata "Марки разделов" (ru-file-menu-xml
    "options\\marka_rz.xml")))
  (if (and (/= data "")(/= data *hole_owner))(_change_owner data))))
;;;-------------------------------------------------------------------------
;;; Локальная функция редактирования владельца
;;;-------------------------------------------------------------------------
(defun _edit_owner (new_owner)
 (if (and (/= new_owner "")(/= new_owner *hole_owner))
 (_change_owner new_owner)))
;;;-------------------------------------------------------------------------
;;; Локальная функция редактирования высоты отверстия
;;;-------------------------------------------------------------------------
(defun _edit_height ()
;; При неверном вводе останется старое значение
 (setq *hole_height (ru-dcl-check-tile-param "ed_hole_height" "Высота"
    50 6000.0 *hole_height 0 "button_do")))
;;;-------------------------------------------------------------------------
;;; Локальная функция редактирования ширины отверстия
;;;-------------------------------------------------------------------------
(defun _edit_width ()
 (setq *hole_width (ru-dcl-check-tile-param "ed_hole_width" "Ширина" 50
   6000.0 *hole_width 0 "button_do")))
;;;-------------------------------------------------------------------------
;;; Локальная функция выбора ширины стены из словаря
;;;-------------------------------------------------------------------------
(defun _select_wall_width (wall_width / data)
 (if (setq data (ru-xml-get-sdata "Типовые стены"
   (ru-file-menu-xml "options\\wall_w.xml")))
   (atof (car (ru-string-to-list data ";"))) wall_width))
;;;-------------------------------------------------------------------------
;;; Локальная функция редактирования ширины стены
;;;-------------------------------------------------------------------------
(defun _edit_wall_width ()
 (setq *wall_width (ru-dcl-check-tile-param "ed_wall_width"
  "Ширина стены" 50 1000.0 *wall_width 0 "button_do")))
;;;-------------------------------------------------------------------------
;;; Локальная функция редактирования отметки отверстия от пола
;;;-------------------------------------------------------------------------
(defun _edit_level ()
```

```
 (setq *level_from_floor (ru-dcl-check-tile-param "ed_level" "Отметка"
   -100000 100000 *level_from_floor 0 "button_do")))
;;;-------------------------------------------------------------------------
;;; Локальная функция выбора отверстия из справочника
;;;-------------------------------------------------------------------------
(defun _select_dim (/ data height width)
 (if (setq data (ru-xml-get-sdata "Типовые отверстия"
   (ru-file-menu-xml "options\\wallhole.xml")))
  (prong
;|
В справочнике часть данных задана с фиксированными размерами отверстий (например,
для стояков и подводок), а часть определяется расчетом в зависимости от диаметра
трубопровода или размеров сторон прямоугольных воздуховодов. На необходимость
выяснения диаметров указывают целочисленные коды, получаемые вместо реальных
размеров отверстий
|;
   (setq data (ru-string-to-list data ";") width (atof (car data))
    height (atof (cadr data)))
   (cond
    ((and (= width 3) (= height 1))
     (if (setq data (ru-xml-get-sdata "Выбор диаметра трубопровода"
        (ru-file-menu-xml "options\\du_lst.xml")))
;; Для труб размер отверстия больше на 100 мм диаметра трубы
      (setq *hole_height
       (+ 100.0 (atof (car (ru-string-to-list data ";")))))))
;; Для воздуховодов размер отверстия больше на 150 мм размера воздуховода
     ((and (= width 2) (= height 1))
;; Прямоугольные
     (if (setq data (ru-xml-get-sdata "Выбор ширины воздуховода"
        (ru-file-menu-xml "options\\vzd_kr.xml")))
       (progn
        (setq *hole_width
          (+ (atof (cadr (ru-string-to-list data ";"))) 150.0))
        (if (setq data (ru-xml-get-sdata "Выбор высоты воздуховода"
           (ru-file-menu-xml "options\\vzd_kr.xml")))
          (setq *hole_height
            (+ (atof (cadr (ru-string-to-list data ";"))) 150.0)))
     )))
     ((and (= width 2) (= height 0))
;; Круглые
      (if (setq data (ru-xml-get-sdata "Выбор диаметра воздуховода"
         (ru-file-menu-xml "options\\vzd_kr.xml")))
        (setq *hole_height
          (+ (atof (cadr (ru-string-to-list data ";"))) 150.0))))
     (T (setq *hole_width width *hole_height height)
)))))
;;;-------------------------------------------------------------------------
;;; Локальная функция пересчета отметки от пола при изменении высоты или
;;; вида отметки
;;;-------------------------------------------------------------------------
(defun _change_level ()
```

```
 (setq *bottom_level
   (cond (*is_axes_level (- *level_from_floor (/ *hole_height 2)))
 (T *level_from_floor))) (princ))
;;;----------------------------------------------------------------------
;;; Локальная функция обработки переключателя вида отметки
;;;----------------------------------------------------------------------
(defun _pick_level (name_what_pick)
 (setq *is_axes_level (= name_what_pick "axes_level")))
;;;----------------------------------------------------------------------
;;; Локальная функция обработки переключателя вида привязки
;;;----------------------------------------------------------------------
(defun _pick_dim (name_what_pick)
 (setq *is_dim_axes (= name_what_pick "dim_axes")))
;;;----------------------------------------------------------------------
;;; Локальная функция справки
;;;----------------------------------------------------------------------
(defun _help (/ err)
 (ru-dlg-view-txt-file (ru-file-help-txt dlg_file) "О программе"
 "Задание на отверстия" (and *ru_developer* (ru-user-may-txt-hlp-edit))))
;;;----------------------------------------------------------------------
;;; Локальная функция тестирования ввода
;;;----------------------------------------------------------------------
(defun _test (/ err)
 (setq err (cond
  ((<= *hole_height 0) "Недопустимая высота отверстия")
  ((<= *hole_width 0) "Недопустимая ширина отверстия")
  ((<= *wall_width 0) "Недопустимая толщина стены")
  ((= *hole_owner "") "Не указана марка")
  (T "")))
 (set_tile "error" err)
 (mode_tile "button_do" (ru-conv-value-to-wordbool (not (= err "")))))
)
;;;----------------------------------------------------------------------
;;; Главная программа
;;;----------------------------------------------------------------------
 (ru-app-begin)
;; Внедрение библиотеки блоков
 (if (ru-block-lib-insert "ru-lib-build" "RU_WALL_HOLE_MARK")
  (progn
   (setq dlg_file "ru_ar_hole_wall" dlg_name "start" what_next 100)
   (if (> (setq DCL_IDX (load_dialog (ru-file-dcl dlg_file))) 0)
    (progn
;; Восстановление параметров
    (_restore)
;; инициализация переменных
    (while (> what_next 0)
     (if (new_dialog dlg_name DCL_IDX)
      (progn
;; Установка данных
      (_set_tiles)
```

```
;; Определение обработчиков событий
      (action_tile "ed_hole_owner" "(_edit_owner $value)(_set_tiles)")
      (action_tile "dic_hole_owner" "(_select_owner)(_set_tiles)")
      (action_tile "ed_hole_height"
                      "(_edit_height)(_change_level)(_set_tiles)")
      (action_tile "get_hole_height" "(done_dialog 2)")
      (action_tile "ed_hole_width" "(_edit_width)(_set_tiles)")
      (action_tile "get_hole_width" "(done_dialog 3)")
      (action_tile "ed_wall_width" "(_edit_wall_width)(_set_tiles)")
      (action_tile "get_wall_width" "(done_dialog 4)")
      (action_tile "dic_wall_width"
       "(setq *wall_width (_select_wall_width *wall_width))(_set_tiles)")
      (action_tile "dim" "(_pick_dim $value)(_set_tiles)")
      (action_tile "level"
                      "(_pick_level $value)(_change_level)(_set_tiles)")
      (action_tile "ed_level" "(_edit_level)(_change_level)(_set_tiles)")
      (action_tile "analog" "(done_dialog 6)")
      (action_tile "dic_hole_dim"
                       "(_select_dim)(_change_level)(_set_tiles)")
      (action_tile "otv_spr" "(done_dialog 5)")
      (action_tile "button_do" "(done_dialog 1)")
      (action_tile "cancel" "(done_dialog 0)")
      (action_tile "help" "(_help)");_ end of action_tile
;; Запуск диалогового окна
      (setq what_next (start_dialog))
      (cond
;; Рисование отверстия
       ((= 1 what_next)(_draw_hole))
;; Указание высоты в рисунке
       ((= 2 what_next)
        (setq *hole_height (ru-conv-unit-to-millimeter (ru-get-dist
          "Высота отверстия" (ru-conv-millimeter-to-unit *hole_height)
           nil))) (_change_level))
;; Указание ширины в рисунке
       ((= 3 what_next)
        (setq *hole_width (ru-conv-unit-to-millimeter (ru-get-dist
          "Ширина отверстия" (ru-conv-millimeter-to-unit *hole_width)
           nil))))
;; Указание ширины стены в рисунке
       ((= 4 what_next)
        (setq *wall_width (ru-conv-unit-to-millimeter (ru-get-dist
          "Ширина стены" (ru-conv-millimeter-to-unit *wall_width)
           nil))))
;; Справка об отверстии
       ((= 5 what_next)(_hole_info))
;; Установка параметров по аналогу
       ((= 6 what_next)(_analog) (_change_level))
;; определено в базовом DCL
       ((= 8 what_next)(ru-dcl-hide-dialog))
      )
     )
```

```
    (progn
     (setq what_next 0) (ru-msg-alert
      (strcat "Не могу показать диалог '" dlg_name "'!"))
   ))) (_save)(unload_dialog DCL_IDX))
   (ru-msg-alert (strcat "Не могу загрузить файл диалога \n'"
    (ru-file-dcl dlg_file) "'!")))))
 (ru-app-end) (princ))
```

Все сложности в программе вызваны необходимостью последующего формирования ведомости отверстий. Каждое отверстие характеризуется строкой, сформированной из значений параметров отверстия функцией `ru-holes-wall-string` (листинг 27.9).

Листинг 27.9. Функция `ru-holes-wall-string`

```
(defun ru-holes-wall-string (hole_width hole_height wall_width bottom_level)
 (strcat
  (vl-princ-to-string (ru-conv-unit-to-millimeter hole_width)) ";"
  (vl-princ-to-string (ru-conv-unit-to-millimeter hole_height)) ";"
  (vl-princ-to-string (ru-conv-unit-to-millimeter wall_width)) ";"
  (vl-princ-to-string (ru-conv-unit-to-millimeter bottom_level))))
```

Функция `ru-holes-wall-hole-dic-dot-pair` (листинг 27.10) создает точечную пару из строки описания и номера отверстия. Необычное расположение номера на втором месте вызвано тем, что нам потребуется поиск номера по строке описания.

Листинг 27.10. Функция `ru-holes-wall-hole-dic-dot-pair`

```
(defun ru-holes-wall-hole-dic-dot-pair (number hole_width hole_height wall_width
bottom_level)
(cons
  (ru-holes-wall-string hole_width hole_height wall_width bottom_level)
  number))
```

Из таких точечных пар формируется список параметров отверстий. При изменении владельца (марки раздела) список сохраняется в словарь рисунка в виде переменной с именем, включающим имя владельца (листинг 27.11), а список отверстий нового владельца читается из словаря (листинг 27.12). Так как в словарь *(см. главу 13)* можно записывать только списки с числом на первом месте в точечной паре, при чтении и записи элементы списка переворачиваются.

Листинг 27.11. Функция `ru-holes-wall-write-dic`

```
(defun ru-holes-wall-write-dic (owner hole_dic_list)
 (ru-dictvar-set-data (strcat "RU_WALL_HOLE_MARK_" owner)
  (ru-list-reverse-assoc hole_dic_list)))
```

Листинг 27.12. Функция `ru-holes-wall-read-dic`

```
(defun ru-holes-wall-read-dic (owner)
 (ru-list-reverse-assoc
  (ru-dictvar-get-data (strcat "RU_WALL_HOLE_MARK_" owner))))
```

Для того чтобы каждое отверстие со своим набором характеристик имело уникальный номер, в момент рисования отверстия производится поиск характеристик в имеющемся списке (листинг 27.13), если такое отверстие не найдено, номер увеличивается на единицу, иначе принимается найденный номер.

Листинг 27.13. Функция `ru-holes-wall-find-hole-number`

```
(defun ru-holes-wall-find-hole-number
       (hole_width hole_height wall_width bottom_level hole_dic_list)
 (car (ru-list-massoc
   (ru-holes-wall-string hole_width hole_height wall_width bottom_level)
       hole_dic_list)))
```

Формирование списка значений атрибутов вставленных блоков отверстий производится функцией, приведенной в листинге 27.14.

Листинг 27.14. Функция `ru-holes-list-wall-holes`

```
(defun ru-holes-list-wall-holes (/ list_holes_blocks result _bottom_level
_hole_height _hole_owner _hole_width _number _wall_width)
(if (setq list_holes_blocks (ru-ss-to-ent-list
  (ssget "_X" (list '(0 . "INSERT") (cons 2 "RU_WALL_HOLE_MARK")))))
 (progn
  (mapcar (function (lambda (ent)
   (if (and (setq
         _hole_width   (ru-block-attrib-by-name ent "WIDTH")
         _hole_height  (ru-block-attrib-by-name ent "HEIGHT")
         _wall_width   (ru-block-attrib-by-name ent "LENGTH")
         _bottom_level (ru-block-attrib-by-name ent "BOTTOM_LEVEL")
         _hole_owner   (ru-block-attrib-by-name ent "OWNER")))
   (setq result (cons
    (list _hole_owner _hole_width _hole_height _wall_width _bottom_level)
                          result)))))
   list_holes_blocks
 )))
;; Удаление повторяющихся элементов
 (ru-list-remove-dublicates result)
)
```

Удаление повторяющихся элементов из списка производится функцией, приведенной в листинге 27.15.

Листинг 27.15. Функция `ru-list-remove-dublicates`

```
(defun ru-list-remove-dublicates (lst / result)
(foreach x lst
  (if (not (member x result))(setq result (cons x result))))
 (reverse result))
```

Конвертирование набора в список примитивов производится функцией, приведенной в листинге 27.16.

Листинг 27.16. Функция `ru-ss-to-ent-list`

```
(defun ru-ss-to-ent-list (selection)
  (if selection
    (vl-remove-if-not
      (function (lambda (x) (= (type x) 'ename)))
      (mapcar 'cadr (ssnamex selection))))
)
```

Проверка корректности введенных данных в поля ввода диалогового окна производится функцией, приведенной в листинге 27.17 (есть и более гибкий вариант проверки — функция `ru-dcl-check-tile-entry`).

Листинг 27.17. Функция `ru-dcl-check-tile-param`

```
(defun ru-dcl-check-tile-param (tile_name txt nmin nmax old dec locked_tile / val
tmp res key)
;;; Проверка правильности ввода в текущее поле диалога
;|Аргументы:
tile_name - имя элемента, в котором проверяется ввод
nmin - минимально допустимое значение
nmax - максимально допустимое значение
old - старое значение
dec - точность отображения чисел
locked_tile - ключ кнопки, которую нужно заблокировать при ошибке
|;
 (if (setq tmp (atof (get_tile tile_name)))
  (progn
   (setq res "") (mode_tile locked_tile 0)
   (if (> tmp nmax) (setq res (strcat txt " " (rtos tmp 2 dec) " больше "
      (rtos nmax 2 dec))))
   (if (< tmp nmin) (setq res (strcat txt " " (rtos tmp 2 dec) " меньше "
      (rtos nmin 2 dec))))
   (if (/= res "")
    (progn
     (mode_tile tile_name 3)(ru-msg-alert res)(setq tmp old)))
   (set_tile tile_name (rtos tmp 2 dec))
))
  tmp
)
```

Исходный текст диалогового окна приведен в листинге 27.18.

Листинг 27.18. Файл ru-holes-wall.dcl

```
// Отверстия в стенах
dcl_settings : default_dcl_settings { audit_level = 0; }
@include "ru_cad_lib.dcl"
get_hole_height: boxed_column {label="Высота";
: row {: edit_box {key = "ed_hole_height";edit_limit = 10;
   edit_width = 6;}
```

```
: button {label = "<";key = "get_hole_height";fixed_width = true;
width = 1;}} spacer; spacer;}
get_hole_width : boxed_column {label="Ширина";
: row {: edit_box {key = "ed_hole_width";edit_limit =10;edit_width = 6;}
 : button {label = "<";key = "get_hole_width";fixed_width = true;
width = 1;}} spacer; spacer;}
get_wall_width:boxed_column {label="Ширина стены";
: row {: edit_box {key = "ed_wall_width"; edit_limit = 10;
edit_width = 6;}
 : button {label = "<";key = "get_wall_width";fixed_width = true;
width = 1;}
: button {label = ">>";key = "dic_wall_width";fixed_width = true;
width = 2;}} spacer; spacer;}
box_dim: boxed_column {label="Проставлять размер до";
: radio_row {key = "dim";: radio_button {label = "Оси";
 key = "dim_axes";value = "1";}: radio_button {label = "Края";
key = "dim_border";value = "0";}} spacer; spacer;}
get_level: column {: edit_box {key = "ed_level";edit_limit = 10;
 edit_width = 6;} spacer; spacer;}
box_level: boxed_row {label="Высота от пола, мм";
get_level;:column {: radio_row {key = "level";
: radio_button {label = "Оси";key = "axes_level";value = "1";}
: radio_button {label = "Низа";key = "bottom_level";value = "0";}}
spacer; spacer;}}
get_analog: boxed_column {label="Взять размеры";
:row {: button {label = "Аналог";key = "analog";fixed_width = true;
width = 6;}: button {label = "СНиП";key = "dic_hole_dim";
fixed_width = true; width = 4;}} spacer; spacer;}
box_hole_dim: boxed_column {label = "Размеры отверстия, мм";
 :concatenation {get_hole_width; get_hole_height; get_wall_width;
get_analog;} spacer; spacer;}
get_hole_owner: boxed_column {label="Владелец отверстия";
:row {: edit_box {key = "ed_hole_owner";edit_limit = 6;}
: button {label = ">>";key = "dic_hole_owner";fixed_width = true;
width = 2;}} spacer; spacer;}
start
:dialog {label="Отверстия в стенах";key = "title";
   box_hole_dim; : row {box_level; box_dim; get_hole_owner;}
   errtile;
 : row {do_exit_hide_help;: button {label = "Чье?"; key = "otv_spr";
fixed_width = true; width = 2;}}}
```

Составление ведомости отверстий в стенах

После того как все отверстия в стенах нарисованы, можно составить и нарисовать ведомость отверстий. Ведомость можно составить и после сведения в один файл слоев, полученных от разных исполнителей — конечно, если отверстия нарисованы нашей программой. Ранее нарисованные ведомости можно просто стереть. Функция создания ведомости отверстий в стенах приведена в листинге 27.19.

Листинг 27.19. Функция `ru-holes-wall-report`

```
(defun ru-holes-wall-report (/ bottom_level first hole_dic_list hole_height
hole_number hole_owner hole_width list_holes pnt tmp wall_width)
(if (ru-yes (strcat "\nКоманда создает ведомость отверстий в стенах."
       "\n\nОтверстия должны быть предварительно проставлены"
       "\nс помощью команды 'Отверстия в стенах'!"
       "\n\nБудем делать"))
  (progn
   (ru-app-begin)
   (if (ru-yes "Нарисовать форму ВЕДОМОСТЬ ОТВЕРСТИЙ В СТЕНАХ")
    (ru-table-draw-with-ask "tab_otvs" nil T))
   (if (setq list_holes (ru-holes-list-wall-holes))
    (progn
     (setq pnt (polar (ru-get-point-reguired
      "Левый нижний угол графы первой строки ведомости отверстий" nil)
       (ru-geom-go-left 0) (ru-conv-millimeter-in-paper-to-unit 4.0))
      first T)
     (foreach hole list_holes
      (setq hole_owner    (nth 0 hole)
            hole_width    (nth 1 hole)
            hole_height   (nth 2 hole)
            wall_width    (nth 3 hole)
            bottom_level  (nth 4 hole)
            hole_dic_list (ru-holes-wall-read-dic hole_owner)
            hole_number   (ru-holes-wall-find-hole-number
                           hole_width hole_height wall_width bottom_level
                           hole_dic_list
                          ))
      (if hole_number
       (setq hole_number (itoa hole_number))
      ;; Для неизвестных отверстий
       (setq hole_number "???")
     )
      (if (not first)
       (if (setq tmp
         (_ru-get-with-default "Левый нижний угол графы следующей записи"
                      "Авто" 'getpoint nil nil nil))
      ;; вернет или указанную точку, или nil, при Enter
       (setq pnt (polar (trans tmp 1 0)
                        (ru-geom-go-left 0)
                        (ru-conv-millimeter-in-paper-to-unit 4.0))))
        (setq first nil
              pnt  (trans pnt 1 0))
     )
     (setq pnt
      (ru-text-draw-in-table pnt
          (list (strcat hole_owner "-" hole_number)
            hole_width hole_height wall_width bottom_level hole_owner)
                (list 20.0 25.0 25.0 25.0 25.0 25.0)))
    ))
```

```
    (ru-msg-alert "Отверстия в стенах не найдены")
  ) (ru-app-end))) (princ)
)
```

После разработки программы рисования отверстий в стенах мы аналогичным образом разрабатываем программу рисования отверстий в перекрытиях (рис. 27.12).

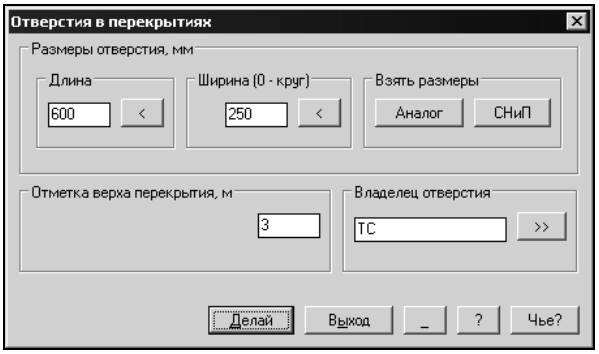


Рис. 27.12. Диалоговое окно программы рисования отверстий в перекрытиях

Отверстия в перекрытиях рисуются по таким же принципам, но с учетом некоторых особенностей:

- ❒ отверстия могут быть круглыми и прямоугольными;
- ❒ требуется задавать угол поворота отверстия;
- ❒ должны быть выполнены две размерные привязки.

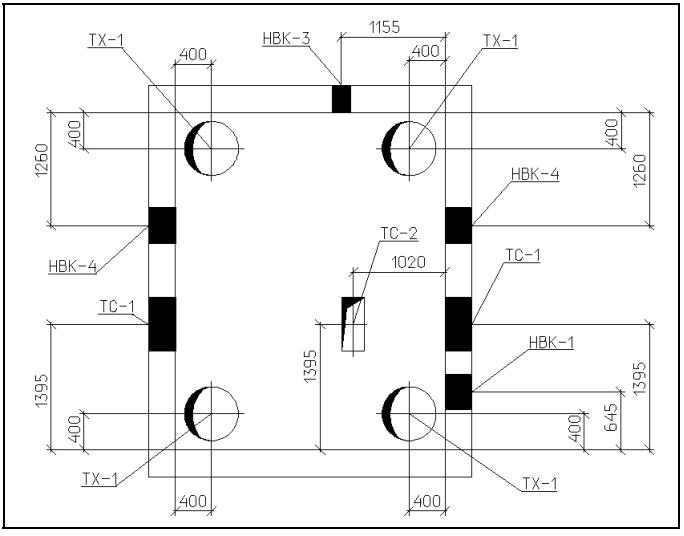


Рис. 27.13. Пример плана отверстий

Результат рисования отверстий показан на рис. 27.13, составления ведомостей отверстий — на рис. 27.14.

ВЕДОМОСТЬ ОТВЕРСТИЙ В СТЕНАХ И ПЕРЕГОРОДКАХ

Марка	Размеры, мм			Отм. низа от 0.00, м	Марка чертежей	Примечание
	Ширина	Высота	Толщ. стены			
НВК-4	400.0	400.0	300.0	400.0	НВК	
НВК-1	400.0	400.0	300.0	1000.0	НВК	
НВК-2	200.0	200.0	300.0	1100.0	НВК	
ТС-1	600.0	300.0	300.0	1800.0	ТС	

ВЕДОМОСТЬ ОТВЕРСТИЙ В ПЕРЕКРЫТИЯХ

Марка	Размеры, мм			Отметка верха перекрытия	Марка чертежей	Примечание
	Длина	Ширина	Диаметр			
ТХ-1			600.0	3.0	ТХ	
ТС-2	600.0	250.0		3.0	ТС	

Рис. 27.14. Пример ведомости отверстий

Резюме

Мы рассмотрели только несколько программ для рисования строительной части. За пределами обзора остались программы для рисования различных проемов, планов и разрезов лестниц, элементов полов, элементов фасадов, выполнение схем и детализировки конструкций, рисование различного оборудования и многое другое.

Большинство таких программ просты для разработки, особенно с учетом возможности применения функций из нашей библиотеки. Основу для многих мы уже рассмотрели, например рисование таблиц, текстов и вставку блоков, а некоторые, такие как использование типовых проектных решений, разберем в последующих главах.

Хочется подчеркнуть необходимость разумного сочетания создания изображений с использованием блоков и с рисованием по рассчитываемым точкам. Нам приходилось встречать программы, в которых рисуется контур железобетонных ферм и плит перекрытия. Это явно неразумно, т. к. геометрия этих изделий не меняется десятилетиями (хотя изменяются серии и маркировка), а программы фактически рисуют всегда одинаковые изображения. В этом случае лучше применить блоки. А вот окна и двери на фасадах мы предпочитаем рисовать программным путем, т. к. их номенклатура очень велика, при небольшом количестве параметров внешнего вида.

ГЛАВА 28

Программы для "генпланистов" и топографов

Программ для рисования топографических и генеральных планов мы разрабатывали очень много, но приведем только несколько характерных примеров. Большинство таких программ не являются сложными для программиста, хотя встречаются исключения, например, составление картограммы земляных работ или профилей коммуникаций. Количество пунктов меню для этих разделов проектов очень велико, но "номенклатура типов программ" ограничена. Специалистам, работающим с планами местности, чаще всего требуется рисование:

- точечных условных знаков, которых насчитывается несколько сотен, но все они могут быть созданы с помощью рассмотренных нами функций вставки блоков;
- линейных изображений (трасс коммуникаций, ограждений), для которых у нас предусмотрено несколько функций рисования трасс;
- дорог различного назначения, которые можно создавать по одному прототипу с отрисовкой одной бровки дороги и созданием второй бровки с использованием подобия;
- различных границ и зон в виде рассмотренных нами контуров и штриховок.

Остальные изображения легко программируются в индивидуальном порядке.

Использование городской системы координат

Работа с планами территорий тесно связана с использованием геоинформационных технологий *(см. главу 34)*, хотя топографы и генпланисты, к сожалению, часто даже не знают о существовании этой отрасли. Но и в этом случае каждый топограф знает, что в САПР топографические планы используются в качестве подосновы для чертежей генплана и сетей, а чертежи генплана и коммуникаций должны отражаться на дежурном плане города. Достичь этих целей можно только при выполнении чертежей в единой системе координат. В каждом приличном населенном пункте такая местная система координат имеется[1]. У крупных предприятий могут быть и собственные местные системы координат, привязка которых к городской системе может быть закрыта.

[1] Одним из немногих исключений является город Москва.

При работе в AutoCAD в качестве местной системы координат следует использовать мировую систему координат (МСК, WCS). Это единственная система, имеющаяся в каждом рисунке. *Все планы местности должны выполняться только в МСК*, какими бы неудобными не казались значения этих координат[1].

За единицу рисунка обычно принимают один метр. Напоминаем, что все наши функции поддерживают систему пропорциональности и позволяют создавать одинаковые на бумаге изображения при работе и с миллиметрами, и с метрами.

Второе важнейшее условие — изображение плана местности никогда не должно переноситься и поворачиваться для удобного размещения на листе. Поворачивайте формат, рисуйте его в ПСК, поворачивайте виды, но никогда не поворачивайте план, иначе его невозможно будет совместить с планом города и другими генпланами.

Третьим важнейшим условием является использование единой системы слоев для всех планов. В нашей системе это обеспечивается с помощью рассмотренного ранее классификатора слоев.

Названия осей координат

В геодезии ось координат, направленная на север, обычно *называется* X, а не Y, как в системе AutoCAD, хотя в некоторых населенных пунктах придерживаются "математического" именования осей. Надо понимать, что X и Y всего лишь символы, обозначающие оси. Как бы мы их не называли, в AutoCAD первое число в списке

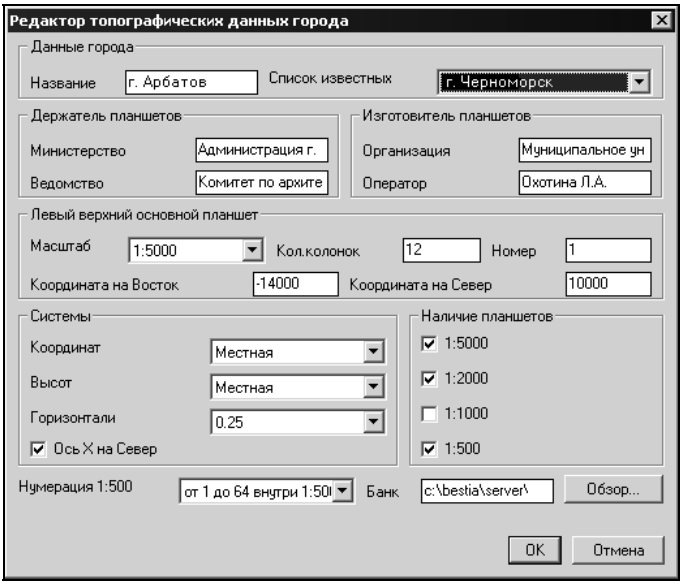


Рис. 28.1. Редактор топографических данных населенного пункта

[1] В некоторых населенных пунктах значения координат в местной системе могут составлять десятки километров.

координат точки соответствует "координате на восток". Правильное обозначение осей имеет значение только в тех случаях, когда оно должно быть написано, например, при выноске координат точки. Так как однозначного наименования осей для всех населенных пунктов нет, направление оси X необходимо уточнять в конкретных проектах.

При работе в полной версии нашей системы, включающей ГИС-модуль, в специальной программе (рис. 28.1) производится выбор населенного пункта, для которого разрабатывается проект и устанавливаются все топографические данные, позволяющие впоследствии легко манипулировать топографическими планшетами. Задерживаться на этой программе мы не будем — она слишком объемна и требует изложения большого материала по системам разграфки топографических планшетов.

Рассмотрим несколько более простых, но эффективных программ.

Работа с координатами объектов

Наибольшее впечатление на "теток", лет по двадцать занимающихся генпланами, производит одна из самых простых программ — выноска координат точки.

Специалисты знают, что координаты углов трасс, участков отвода, границ красных линий и т. п. должны быть обозначены с помощью специальной выноски. При традиционном проектировании определение координат производится занудными графоаналитическими методами, требующими большой аккуратности и точности. В AutoCAD координаты любой точки всегда известны, остается их нарисовать в требуемом виде (рис. 28.2).

Текст программы, как всегда, прост (листинг 28.1).

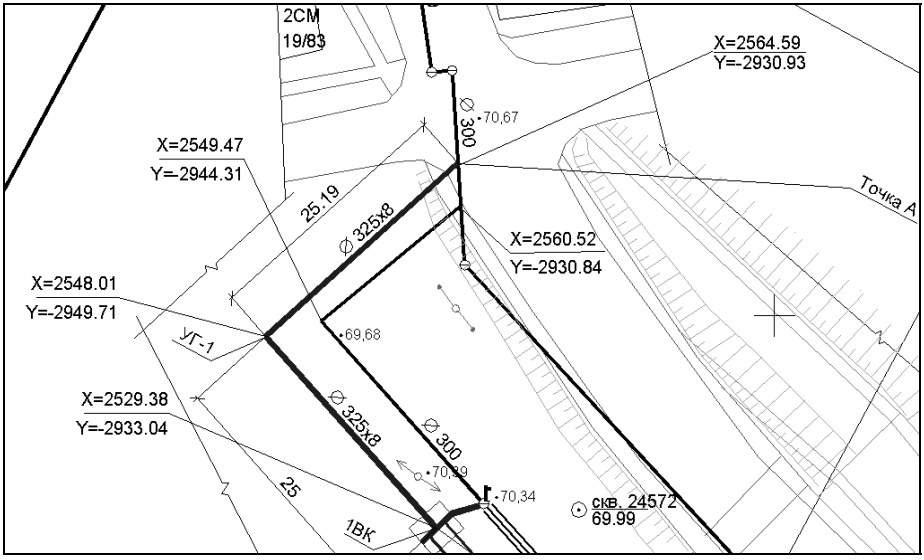


Рис. 28.2. Выноски координат узлов трассы

Листинг 28.1. Файл ru_topo_coord_leader.lsp

```
(defun START (/ start_pnt txt_x txt_y x x_north y)
 (ru-app-begin)
 (setq x_north (ru-conv-str-to-bool
                 (ru-ini-read-default NIL "Setup" "X_axis_North" "1")))
 (while
   (setq start_pnt
        (ru-get-point-or-exit "Укажи точку для выноски координат" nil))
    (setq x (rtos (ru-conv-unit-to-meter (car start_pnt)) 2 2)
          y (rtos (ru-conv-unit-to-meter (cadr start_pnt)) 2 2))
    (if x_north
     (setq txt_x (strcat "X=" y) txt_y (strcat "Y=" x))
     (setq txt_y (strcat "Y=" y) txt_x (strcat "X=" x)))
     (ru-draw-leader-and-two-string txt_x txt_y start_pnt)
)
 (ru-app-end) (princ))
(START)
```

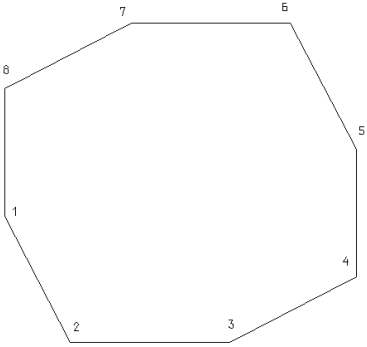
Рисование выноски с двумя строками, применяемое во многих программах, выполняется функцией, приведенной в листинге 28.2.

Листинг 28.2. Функция ru-draw-leader-and-two-string

```
(defun ru-draw-leader-and-two-string (txt1 txt2 start_pnt / align end_pnt len_txt
lst_txt result y)
 (setq y 0.6)
 (if (not start_pnt)
   (setq start_pnt (ru-get-point-or-exit "Начало выноски" nil)))
 (if start_pnt
   (progn
    (setq end_pnt  (ru-get-point-reguired "Конец выноски" start_pnt)
             align   (ru-text-end-leader-align (angle start_pnt end_pnt))
             lst_txt (textbox (list (cons 1 txt1)))
             len_txt (- (car (cadr lst_txt)) (car (car lst_txt))))
    (ru-line-add-multi (list  start_pnt end_pnt
      (if (zerop align)
         (polar end_pnt 0 len_txt)
         (polar end_pnt (ru-geom-go-back 0) len_txt))) nil 0 nil)
    (ru-text-add txt1 (polar end_pnt (ru-geom-go-left 0)
      (* y (ru-normal-text-height))) (ru-normal-text-height) 0 align)
    (ru-text-add txt2 (polar end_pnt (ru-geom-go-right 0)
      (+ (* y (ru-normal-text-height)) (ru-normal-text-height)))
      (ru-normal-text-height) 0 align)
     (setq result t)
  )
)
 result
)
```

Ведомость координат

Еще одна эффектная программа, значительно облегчающая работу многих специалистов — ведомость координат полигонов или трасс (рис. 28.3). Она производит особое впечатление при большом количестве вершин полигона, например, при составлении ведомости координат городской черты. Такой работой, за приличные деньги, может заниматься целый институт. После того как линия границы согласована и начерчена, значительную часть работы будет составлять расчет координат узловых точек, вычерчивание и заполнение ведомости.



ВЕДОМОСТЬ КООРДИНАТ

Узловые точки						Отрезки между точками	
Начальная			Конечная				
Обозн.	Координаты		Обозн.	Координаты		Азимут	Расстояние
	Восток	Север		Восток	Север		
ОБЪЕКТ – Земельный участок							
1	-2789.84	2644.03	2	-2779.62	2624.34	Ю 27°25'18" В	22.19
2	-2779.62	2624.34	3	-2754.79	2624.34	В	24.83
3	-2754.79	2624.34	4	-2735.09	2634.56	С 62°34'42" В	22.19
4	-2735.09	2634.56	5	-2735.09	2654.41	С	19.85
5	-2735.09	2654.41	6	-2745.31	2674.1	С 27°25'18" З	22.19
6	-2745.31	2674.1	7	-2770.14	2674.1	З	24.83
7	-2770.14	2674.1	8	-2789.84	2663.88	Ю 62°34'42" З	22.19
8	-2789.84	2663.88	1	-2789.84	2644.03	Ю	19.85

Площадь 2321.8908 м2

Рис. 28.3. Ведомость координат

В функции составления ведомости (листинг 28.3) единственной сложностью является кропотливое высчитывание координат точек таблицы и текстов. При этом, как всегда, используется очень много библиотечных функций. В аргументах функции передаются название объекта, префикс для номеров точек, максимальная высота таблицы. Если точек много, то таблица после достижения максимальной высоты продолжается вправо.

Листинг 28.3. Функция `ru-table-coords`

```
(defun ru-table-coords (header points_prefix max_table_height / area closed
curr_point first_point lst_points lst_points_txt next_point perimeter pline_ent
point_count point_number point_txt last_start_point next_end_pnt next_start_pnt
row_height _draw_point_txt _make_table _draw_table_line  _draw_table_header
_draw_table_grid)
;;; Пример: (ru-table-coords "Земельный участок" "Т." 50)
;;; ------------------ Расчерчивание таблицы -------------------------
  (defun _draw_table_grid (start_pnt end_pnt down_dist / next_start_pnt
next_end_pnt list_columns_width)
;; Левая черта
 (ru-line-add start_pnt end_pnt 0 nil)
 (setq list_columns_width
   (list 20.0 20.0 20.0 20.0 20.0 20.0 35.0 30.0)
    next_end_pnt       end_pnt
    next_start_pnt     (polar start_pnt (ru-geom-go-right 0) down_dist))
```

```
;; Вертикальные линии
  (mapcar (function (lambda (width / dist)
     (setq dist           (ru-conv-millimeter-in-paper-to-unit width)
         next_start_pnt (polar next_start_pnt 0 dist)
        next_end_pnt   (polar next_end_pnt 0 dist))
     (ru-line-add next_start_pnt next_end_pnt 0 nil)
              )) list_columns_width)
;; Правая черта
  (ru-line-add
      (polar start_pnt 0 (ru-conv-millimeter-in-paper-to-unit 185.0))
      (polar end_pnt 0 (ru-conv-millimeter-in-paper-to-unit 185.0))
      0 nil)
 )
;;; ---------------- Шапка и заголовок ----------------------------
 (defun _draw_table_header (pnt header / scale next_line_start_pnt)
   (setq scale (ru-conv-millimeter-to-unit (ru-scale-current-space))
         row_height (ru-normal-table-row-height))
  (ru-block-insert-obj (ru-file-set-ext (ru-file-table "vedkrd") ".dwg")
     pnt scale scale 1 0)
  (setq pnt (polar pnt (ru-geom-go-right 0)
   (ru-conv-millimeter-in-paper-to-unit 25.0))
       next_line_start_pnt (polar pnt (ru-geom-go-right 0) row_height))
  (ru-text-add (strcat "%%U" "ОБЪЕКТ - " Header)
      (polar (polar pnt 0 (ru-conv-millimeter-in-paper-to-unit 5))
      (ru-geom-go-right 0)
      (- row_height (ru-conv-millimeter-in-paper-to-unit 2)))
      (ru-normal-text-height) 0 NIL)
;; Линия после заголовка
    (ru-line-add next_line_start_pnt (polar next_line_start_pnt 0
            (ru-conv-millimeter-in-paper-to-unit 185.0)) 0 nil)
    (ru-geom-txt-down-line next_line_start_pnt)
)
;;; ------------------ Строка таблицы -------------------------
(defun _draw_table_line (left_row_point  start_pnt end_pnt txt_start_pnt
txt_end_pnt / azimut length_str txt_end_pnt_x   txt_end_pnt_y)
 (if end_pnt
    (setq txt_end_pnt_x (rtos (car end_pnt) 2 2)
        txt_end_pnt_y (rtos (cadr end_pnt) 2 2)
        length_str    (rtos (distance start_pnt end_pnt) 2 2)
        azimut (ru-conv-angle-to-str (angle start_pnt end_pnt) 4 4 "%%d")
     )
    (setq txt_end_pnt_x "" txt_end_pnt_y "" length_str "" azimut "")
)
 (ru-text-draw-in-table (polar left_row_point (ru-geom-go-left 0)
            (ru-conv-millimeter-in-paper-to-unit 2.0))
   (list
    txt_start_pnt (rtos (car start_pnt) 2 2) (rtos (cadr start_pnt) 2 2)
    txt_end_pnt txt_end_pnt_x txt_end_pnt_y azimut length_str
   )
   (list 20.0 20.0 20.0 20.0 20.0 20.0 35.0 30.0))
 (ru-line-add left_row_point (polar left_row_point 0
            (ru-conv-millimeter-in-paper-to-unit 185.0)) 0 nil)
```

```
 (ru-geom-txt-down-line left_row_point)
)
;;;------ Формирование таблицы --------------------------
(defun _make_table (lst_points lst_names_points header Area perimerer
max_table_height Polygon / end_pnt go_up left_pnt left_top_pnt n name2 points_count
row_height txt_pnt)
 (setq
   left_top_pnt (ru-get-point-reguired
    "Левый верхний угол шапки ведомости координат" nil)
   left_pnt (_draw_table_header left_top_pnt header)
   n 0 points_count (length lst_points)
   row_height (ru-normal-table-row-height)
   go_up (ru-geom-go-left 0))
 (repeat points_count
   (if (< n (1- points_count))
    (setq end_pnt (nth (1+ n)
       lst_points) name2 (nth (1+ n) lst_names_points))
    (if Polygon (setq end_pnt (nth 0 lst_points)
                name2   (nth 0 lst_names_points))
          (setq end_pnt nil name2 "")))
;; проверка на превышение высоты
   (if (> (distance left_top_pnt left_pnt) max_table_height)
    (progn
      (_draw_table_grid
        (polar left_top_pnt (ru-geom-go-right 0)
                   (ru-conv-millimeter-in-paper-to-unit 25))
      (setq left_pnt (polar left_pnt go_up row_height)) row_height)
      (setq left_top_pnt (polar left_top_pnt 0
               (ru-conv-millimeter-in-paper-to-unit 190))
            left_pnt (_draw_table_header left_top_pnt
                              (strcat header " (продолжение)")))
   )
 )
  (setq left_pnt (_draw_table_line left_pnt (nth n lst_points) end_pnt
                       (nth n lst_names_points) name2) n (1+ n))
);_ end of repeat
 (_draw_table_grid (polar left_top_pnt (ru-geom-go-right 0)
             (ru-conv-millimeter-in-paper-to-unit 25))
      (setq left_pnt (polar left_pnt go_up row_height))
      row_height)
 (setq left_pnt (polar left_pnt (ru-geom-go-right 0) row_height))
 (if Polygon
  (progn
   (ru-text-add "Площадь " (setq txt_pnt (polar (polar left_pnt 0
                (ru-conv-millimeter-in-paper-to-unit 120)) go_up
               (ru-conv-millimeter-in-paper-to-unit 2))
          (ru-normal-text-height) 0 nil)
   (ru-text-add (strcat (rtos (* Area (ru-conv-unit-to-meter 1)
                 (ru-conv-unit-to-meter 1)) 2 4) " м2")
     (polar txt_pnt 0 (ru-conv-millimeter-in-paper-to-unit 35))
     (ru-normal-text-height) 0 nil)
 )
```

```
  (progn
   (ru-text-add "Длина трассы " (setq txt_pnt (polar (polar left_pnt 0
              (ru-conv-millimeter-in-paper-to-unit 120)) go_up
                 (ru-conv-millimeter-in-paper-to-unit 2)))
          (ru-normal-text-height) 0 nil)
   (ru-text-add (strcat (rtos perimerer 2 4) " " (ru-unit-name))
          (polar txt_pnt 0 (ru-conv-millimeter-in-paper-to-unit 35))
          (ru-normal-text-height) 0 nil)
 )
)
)
;;;------------------ Подписывание точки на плане --------------------
(defun _draw_point_txt (txt pt1 pt2 / angle_rad text_pnt txt_height)
 (setq angle_rad  (angle pt1 pt2) txt_height (ru-normal-text-height)
          text_pnt   (polar pt1 angle_rad txt_height))
 (ru-text-add txt
  (polar (if (and (> angle_rad 1.5708) (<= angle_rad 4.71239))
          (polar text_pnt (ru-geom-go-right angle_rad) txt_height)
          (polar text_pnt (ru-geom-go-left angle_rad) txt_height)
        )
        (ru-geom-go-left 0) (* 0.5 txt_height)) txt_height 0
      (ru-text-end-leader-align (angle pt2 pt1)))
)
;;; ---- Основная функция ------------------------------------------
(if (setq pline_ent (ru-get-entsel-by-type
       "Выбери  полилинию для ведомости координат" "Это не  ПОЛИЛИНИЯ"
                        (list "LWPOLYLINE") T))
  (progn
   (setq lst_points       (ru-pline-list-vertex (car pline_ent))
         closed           (cadr lst_points)
         lst_points       (car lst_points)
         Area             (ru-geom-area-ent (car pline_ent))
         point_number 0 perimeter 0 point_count (length lst_points)
         first_point (nth 0 lst_points) last_start_point first_point)
   (repeat point_count
     (setq curr_point   (nth point_number lst_points)
           point_number (1+ point_number)
           point_txt    (strcat points_prefix (itoa point_number)))
     (if (< point_number point_count)
       (progn
         (setq next_point (nth point_number lst_points)
              perimeter  (+ perimeter (distance curr_point next_point)))
         (_draw_point_txt point_txt curr_point next_point)
      )
       (progn
         (_draw_point_txt point_txt curr_point last_start_point)
        (if closed
         (setq perimeter (+ perimeter (distance next_point first_point)))
       )
      )
     )
```

```
      (setq last_start_point curr_point
             lst_points_txt (cons point_txt lst_points_txt))
  );_ end of repeat
   (_make_table lst_points (reverse lst_points_txt) Header Area perimeter
       max_table_height Closed)
 )
)(princ))
```

Несколько способов рисования откосов

Рисование откосов — тема, являющаяся "пробным камнем" для любой графической системы. Попробуйте нарисовать откос по "советским" стандартам во многих популярных картографических системах — в лучшем случае найдется возможность применить тип линии с чередующимися штрихами постоянной длины. Откосы для мелкомасштабных планов, где не имеет значения размер штрихов, можно выполнять специальным типом линии и в AutoCAD *(см. главу 7)*, но нас интересует стандартное изображение для крупномасштабных планов (рис. 28.4).

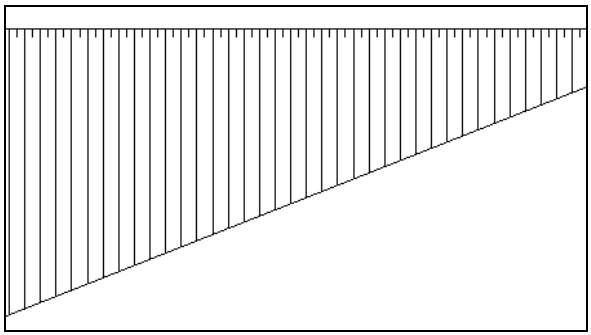


Рис. 28.4. Топографическое условное изображение откоса

Графические условные изображения откосов устанавливаются *ГОСТ 21.108-78 Условные графические изображения и обозначения на чертежах генеральных планов и транспорта* и условными знаками для топографических планов[1].

В общем виде для более наглядного выражения направления ската у верха кромок откосов наносятся штрихи *перпендикулярно горизонталям*[2]. Расстояние между длинными штрихами 3—4 мм, между короткими и длинными 1.5—2 мм. В стандарте указано только расстояние между штрихами, но умалчивается о длине коротких штрихов, а в примерах изображения рельефа короткие штрихи показаны и до середины склона (рис 28.5), и такой же длины, как и расстояние между штрихами.

Условные знаки для топографических планов более конкретны — длина коротких штрихов должна быть равна расстоянию между штрихами. В учебной литературе

[1] Условные знаки для топографических планов масштабов 1:5000, 1:2000, 1:1000, 1:500. — М.: Картгеоцентр — Геодезиздат, 2000. — 286 с.: ил. ISBN 5-86066-046-4.

[2] В ГОСТ 21.108-78 такого выражения нет, это цитата из учебника по строительному черчению.

встречаются различные варианты. Для нас этот мелкий вопрос очень важен, т. к. определяет весь алгоритм программы. Мы будем придерживаться правила — *расстояние между короткими и длинными штрихами и длину коротких штрихов принимаем 2 мм на бумаге*. Мы будем также придерживаться наименований — *бровка* для верхней кромки откоса и *подошва* для нижней.

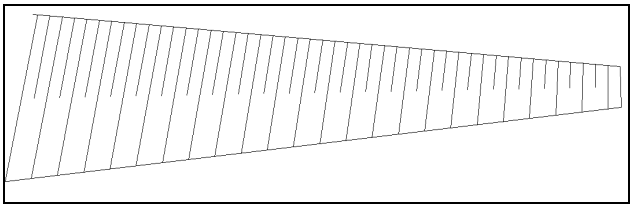


Рис. 28.5. Вариант изображения откоса

Технология рисования зависит от способа определения бровки и подошвы. При топографическом черчении откос обычно "скалывается" с оригинала. Как были рассчитаны кромки и их пересечения, оператора не интересует, ему нужно просто получить такое же, но более аккуратное изображение, не исказив обстановку. В этом случае удобно автоматически выполнять штриховку откоса сразу после нанесения линий бровки и подошвы. При проектировании рельефа кромки и их пересечения строят по правилам начертательной геометрии, и построением таких изображений мучили всех студентов строительных вузов. Разумеется, такие построения очень просто делать в системе AutoCAD, а штриховку откосов лучше вынести в отдельную программу. Этот вариант является универсальным, и его мы примем за основу. Штриховка будет выполняться между примитивами, указанными в качестве бровки и подошвы.

Разбивка кромок

Штрихи рисуются от бровки в направлении подошвы. Разбить бровку на равные участки можно командами MEASURE (РАЗМЕТИТЬ) и DIVIDE (ПОДЕЛИТЬ). Здесь возникает вопрос о том, "кто кому перпендикулярен". Штрихи должны быть якобы "перпендикулярны[1]" горизонтали, но и бровка, и подошва не являются горизонталями (линиями с одинаковыми высотными отметками) — это линии резкого перелома рельефа. Обычно штрихи строят по нормали к бровке, но это не всегда возможно. Рассмотрим классический учебный пример построения откосов (рис. 28.6). В этом варианте штрихи построены по нормали к бровке откоса. Для участков, у которых кромки формируются прямыми или равномерно изогнутыми кривыми, все выглядит прилично, а вот "зуб" в нижней части построен безобразно.

Небольшая правка нескольких штрихов позволяет получить требуемое изображение (рис. 28.7). Именно таким образом были нарисованы откосы на сотнях топографических планшетах с использованием системы BestIA — быстрое рисование основных изображений и мелкая правка отдельных неверных деталей.

---

[1] На нашем, "автокадовском" языке — построены по нормали к кривой.

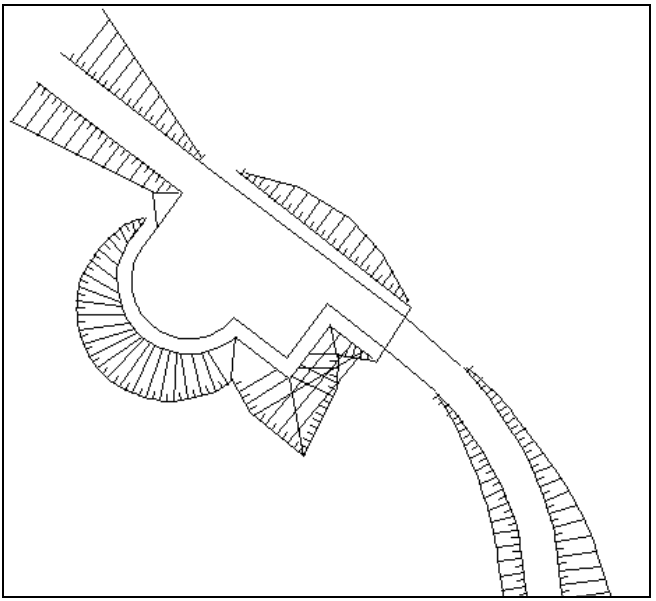


Рис. 28.6. Откосы со штрихами, нормальными бровке

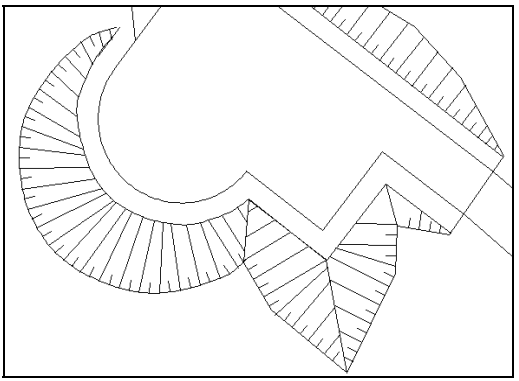


Рис. 28.7. Исправленное изображение со штрихами, нормальными бровке

В этой программе реализован следующий алгоритм (назовем его *метод 1*):

- бровка разбивается с шагом 2 мм на бумаге (блоки длиной 2 мм на бумаге вставляются "перпендикулярно" к бровке с помощью команды MEASURE (РАЗМЕТИТЬ));
- четные блоки-штрихи расчленяются и удлиняются до подошвы.

На мелких деталях алгоритма останавливаться не будем, отметим только, что качество изображения зависит от длины и извилистости подошвы — не все штрихи могут быть удлинены до подошвы, а некоторые могут пересекаться между собой, что недопустимо.

Второй вариант штриховки откоса (рис. 28.8) работает по другому алгоритму (назовем его *метод 2*):

- бровка разбивается точками с шагом 2 мм на бумаге с помощью команды MEASURE (РАЗМЕТИТЬ);
- подошва разбивается точками на такое же количество частей с помощью команды DIVIDE (ПОДЕЛИТЬ);
- координаты точек бровки и подошвы запоминаются, а сами точки удаляются;
- по известным координатам рисуются отрезками длинные и короткие штрихи.

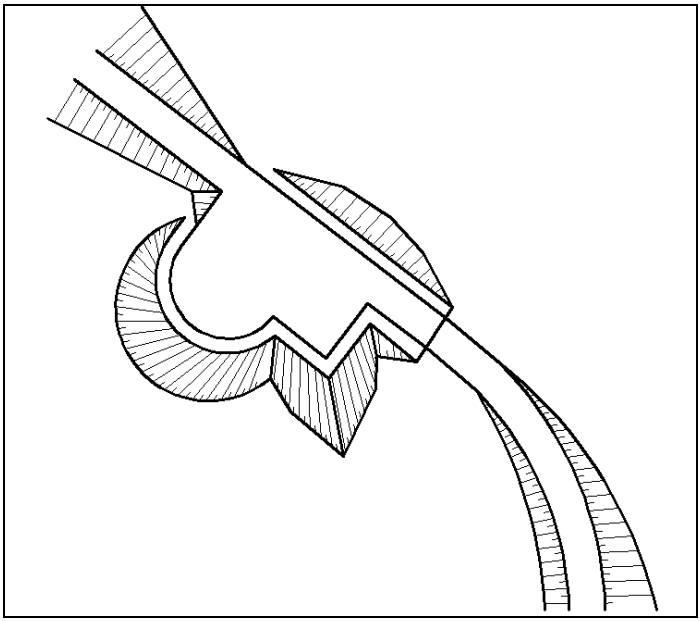


Рис. 28.8. Откос с делением бровки и подошвы на равное количество частей

Судя по рис. 28.8, качество изображения получается лучше, но "перпендикулярность" штрихов к бровке может быть получена только при параллельных кромках. Думаем, что этим можно пренебречь. В реальных условиях изображения будут удовлетворять любых критиков, хотя возможны и неожиданные варианты (рис. 28.9—28.11).

Результаты рисования по методу 2 зависят от того, как система AutoCAD начинает разбивку замкнутых объектов, используемых в качестве подошвы, а также от направления линий. Особое внимание следует уделять варианту с пересечением бровки и подошвы — в этом случае возможно соединение не так, как думал пользователь (рис. 28.11).

Итак, изрядно разобравшись с теорией, мы легко напишем программу рисования откоса по второму методу (листинг 28.4).

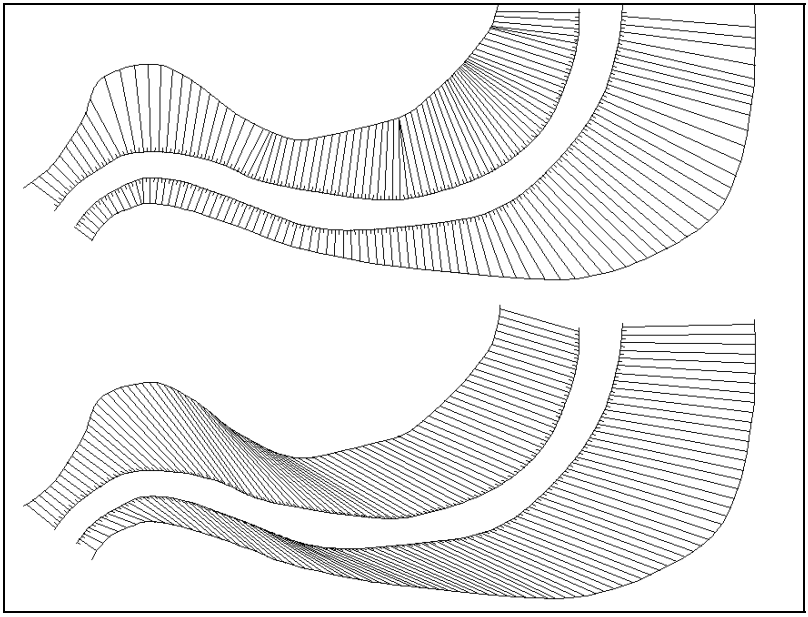


Рис. 28.9. Сравнение вариантов рисования: метод 1 вверху, метод 2 внизу

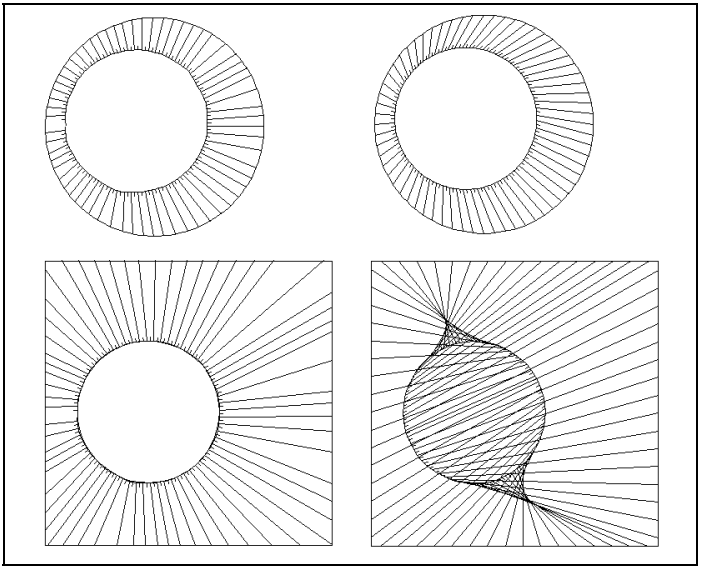


Рис. 28.10. Сравнение вариантов рисования: метод 1 слева, метод 2 справа

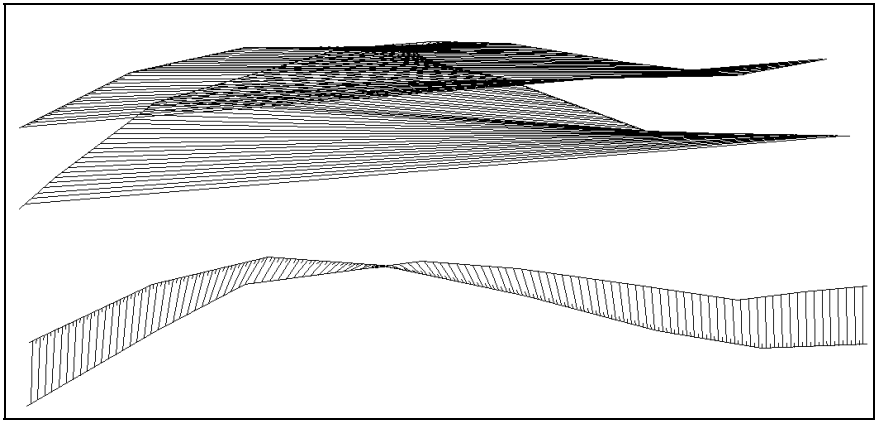


Рис. 28.11. Влияние направления линий на изображение откоса

Листинг 28.4. Файл ru_topo_slope.lsp

```
(defun START (/ bottom_point bottom_points_list_ents dist ent_bottom ent_top
last_ent length_berg long n ss_bottom_points ss_top_points top_point
top_points_list_ents)
(ru-app-begin)
;;; Размер верхних коротких штрихов 2 мм на бумаге
 (setq length_berg (ru-conv-millimeter-in-paper-to-unit 2.0))
 (while
  (setq ent_top (ru-get-entsel-by-type "Выбери линию бровки"
        "Это нельзя использовать в качестве бровки"
       (list "LINE" "LWPOLYLINE" "POLYLINE" "ARC" "CIRCLE" "SPLINE") T))
  (if (setq ent_bottom (ru-get-entsel-by-type "Выбери линию подошвы"
                        "Это нельзя использовать в качестве подошвы"
       (list "LINE" "POLYLINE" "LWPOLYLINE" "ARC" "CIRCLE" "SPLINE") T))
    (progn
      (setq last_ent (entlast) ent_top (car ent_top)
        ent_bottom (car ent_bottom))
       (vl-cmdf "_.MEASURE" ent_top length_berg)
      (setq ss_top_points (ru-ss-select-after-ent last_ent)
                    last_ent (entlast))
      (vl-cmdf "_.DIVIDE" ent_bottom (1+ (sslength ss_top_points)))
      (setq ss_bottom_points (ru-ss-select-after-ent last_ent)
          n 0 top_points_list_ents (ru-ss-to-ent-list ss_top_points)
          bottom_points_list_ents (ru-ss-to-ent-list ss_bottom_points))
     (if (inters (cdr (assoc 10 (entget (car top_points_list_ents))))
                 (cdr (assoc 10 (entget (car bottom_points_list_ents))))
                 (cdr (assoc 10 (entget (last top_points_list_ents))))
                 (cdr (assoc 10 (entget (last bottom_points_list_ents))))
                         T)
         (setq bottom_points_list_ents
            (reverse bottom_points_list_ents)))
```

```
      (mapcar (function (lambda (top_pnt_ent / bottom_pnt_ent)
         (setq top_point (cdr (assoc 10 (entget top_pnt_ent)))
               bottom_pnt_ent (nth n bottom_points_list_ents)
               bottom_point   (cdr (assoc 10 (entget bottom_pnt_ent)))
               dist           (distance top_point bottom_point)
                n              (1+ n)
                long           (not long))
         (entdel top_pnt_ent) (entdel bottom_pnt_ent)
         (if (not long)
          (setq bottom_point (polar top_point
                  (angle top_point bottom_point)
                  (if (< dist length_berg) (* dist 0.5) length_berg))))
         (ru-line-add top_point bottom_point 0 nil)))
         top_points_list_ents
       );_ end of mapcar
     );_ end of progn
    );_ end of if
  );_ end of if
  (ru-app-end)
 (princ)
)
```

Рисование трасс

Теперь разработаем несколько программ для рисования трасс различных коммуникаций. Начнем с воздушных электролиний. В "голой" системе AutoCAD рисовать паутину ЛЭП со столбами сущее мучение — множество столбов с "зазубренными" наконечниками направлений трасс. Столбы бывают из разного материала, ответвлений от столбов разное количество, и все надо сделать аккуратно и единообразно, поэтому часто программа рисования ЛЭП разрабатывается в первую очередь. Опоры ЛЭП мы, конечно, будем создавать из блоков. Сначала включим в библиотеку топографических блоков несколько блоков опор из разных материалов для низкого (одна "зазубрина") и высокого (две "зазубрины") напряжения (рис. 28.12). Диаметр окружности всех опор одна единица.

Для рисования ЛЭП очень удобна функция `ru-trass-draw-first-prev-sample` *(см. главу 22)*, которой требуется передать один аргумент — имя функции рисования сегмента. Мы можем сразу написать универсальную функцию рисования любых ЛЭП (листинг 28.5).

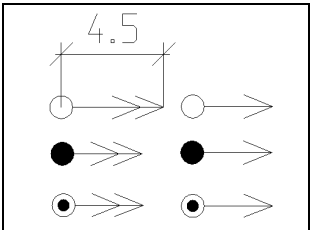


Рис. 28.12. Блоки опор линий электропередачи

Листинг 28.5. Функция ru-draw-electro-pole

```
(defun ru-draw-electro-pole (funct)
  (ru-trass-draw-first-prev-sample (read funct))
(princ)
)
```

Эта функция может показаться ничтожной, но обратите внимание на то, что она передает основной функции свой единственный аргумент, прочитанный функцией read. Такой прием позволяет вызвать функцию рисования ЛЭП из меню системы AutoCAD или из XML-меню посредством макросов (листинг 28.6).

Листинг 28.6. Фрагмент XML-меню рисования ЛЭП

```
<item name='Воздушные линии' macro=''>
  <item name='По опорам' macro=''>
   <item name='Деревянные опоры' macro=''>
    <item name='Низкое  напряжение' image='TOPO\SETI\VL_POLE_W_NN.GIF'
       comment='Рисование ВЛ по столбам '
        macro='(ru-draw-electro-pole "СТОЛБ_ДЕРЕВО_НВ")'/>
    <item name='Высокое напряжение' image='TOPO\SETI\VL_POLE_W_VN.GIF'
       comment='Рисование ВЛ по столбам '
       macro='(ru-draw-electro-pole "СТОЛБ_ДЕРЕВО_ВВ")'/>
   </item>
   <item name='Железобетонные опоры' macro=''>
     <item name='Низкое  напряжение' image='TOPO\SETI\VL_POLE_GB_NN.GIF'
       comment='Рисование ВЛ по столбам '
       macro='(ru-draw-electro-pole "СТОЛБ_БЕТОН_НВ_ТОПО")'/>
     <item name='Высокое  напряжение' image='TOPO\SETI\VL_POLE_GB_VN.GIF'
       comment='Рисование ВЛ по столбам '
       macro='(ru-draw-electro-pole "СТОЛБ_БЕТОН_ВВ_ТОПО")'/>
   </item>
  </item>
  <item name='Линия по опорам' macro=''>
   <item name='Деревянные опоры' macro=''>
     <item name='Низкое  напряжение' image='TOPO\SETI\VL_POLE_W_NN_L.GIF'
         comment='Рисование ВЛ по столбам '
         macro='(ru-draw-electro-pole "СТОЛБ_ДЕРЕВО_НВ_ЛИНИЯ")'/>
     <item name='Высокое напряжение' image='TOPO\SETI\VL_POLE_W_VN_L.GIF'
         comment='Рисование ВЛ по столбам '
          macro='(ru-draw-electro-pole "СТОЛБ_ДЕРЕВО_ВВ_ЛИНИЯ")'/>
   </item>
 </item>
</item>
```

Итак, мы передаем имя *функции рисования сегмента в виде строки*, а в результате для *выполнения вызывается функция с таким же именем*. Попробуйте проделать такой фокус в программах, написанных на других языках программирования!

Меню для работы с ЛЭП будет выглядеть так, как показано на рис. 28.13.

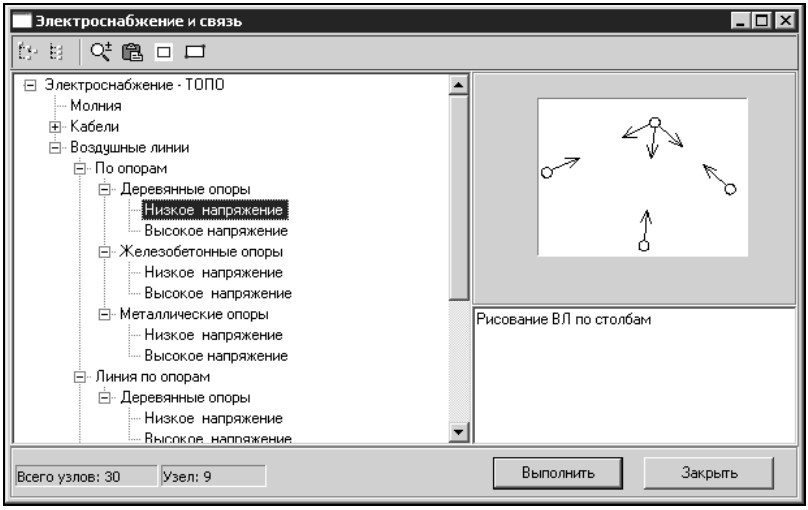


Рис. 28.13. Меню рисования ЛЭП во время выбора

Напишем для примера пару таких функций (листинги 28.7 и 28.8). Реально таких функций могут быть десятки, и не только для ЛЭП, а для любых изображений, имеющих точку начала и конца.

Листинг 28.7. Функция СТОЛБ_ДЕРЕВО_НВ

```
(defun СТОЛБ_ДЕРЕВО_НВ[1] (start_point end_point)
  (ru-draw-electro-pole-segm
       start_point end_point "СТОЛБ_ДЕРЕВО_НВ" 1.5 NIL 0.5)
)
```

Листинг 28.8. Функция СТОЛБ_ДЕРЕВО_НВ_ЛИНИЯ

```
(defun СТОЛБ_ДЕРЕВО_НВ_ЛИНИЯ (start_point end_point)
  (ru-draw-electro-pole-segm
       start_point end_point "СТОЛБ_ДЕРЕВО_НВ" 1.5 "CONTINUOUS" 0.5)
)
```

Все подобные функции вызывают еще одну, которая и рисует сегмент (листинг 28.9).

Листинг 28.9. Функция ru-draw-electro-pole-segm

```
(defun ru-draw-electro-pole-segm (start_point end_point block_name   block_scale
ltype line_dist / dist scale)
```

[1] Русские имена функций и блоков, противоречащие концепции нашей системы, оставлены временно — мы их переименуем в конце разработки в пакетном режиме. Кроме того, мы демонстрируем, что в LISP можно применять и такие имена.

```
;; Внедрение блока из библиотеки
  (if (ru-block-lib-insert "ru-lib-topo" block_name)
    (progn
      (setq *ru_last_end_point* start_point
            scale (ru-conv-millimeter-in-paper-to-unit block_scale)
            dist (ru-conv-millimeter-in-paper-to-unit line_dist))
;; Вставка начального блока
          (ru-block-insert-obj block_name start_point scale scale scale
            (angle start_point end_point))
;; Вставка конечного блока
          (ru-block-insert-obj block_name end_point scale scale scale
            (angle end_point start_point))
;; Если задан тип линии, рисование "провода"
      (if ltype (ru-line-add
          (polar start_point (angle start_point end_point) dist)
          (polar end_point (angle end_point start_point) dist) 0 ltype))
      (setq *ru_last_start_point* end_point)
   )
 )
)
```

В результате работы этих функций могут быть нарисованы ЛЭП, подобные показанным на рис. 28.14.

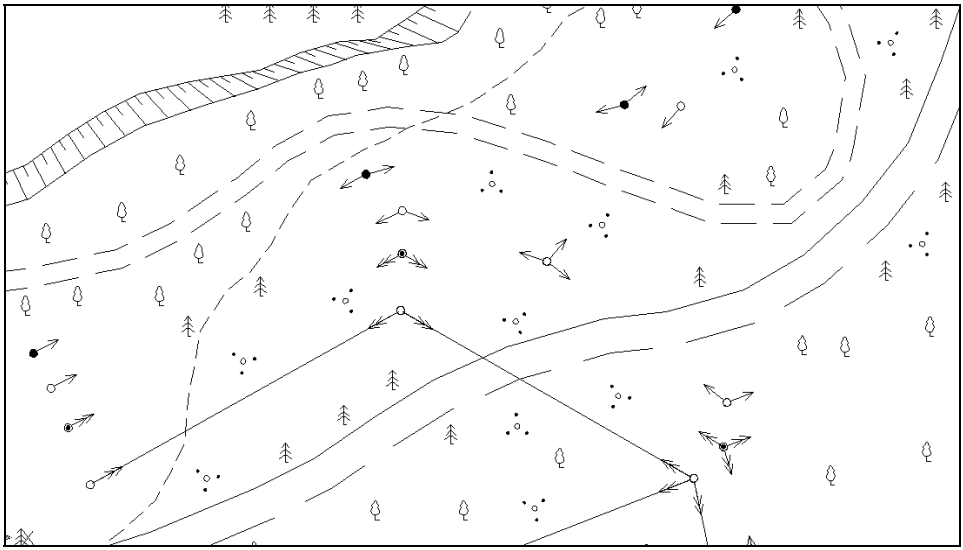


Рис. 28.14. Топографический план с ЛЭП, дорогами и растительностью

На рис. 28.14 заметны еще несколько изображений. Условные знаки деревьев и кустарников нарисованы с помощью функции многократной вставки блока, с откосом мы уже знакомы, остановимся теперь на дорогах и тропинках.

Рисование дорог

Рисовать топографические дороги можно и обычными примитивами. Однако возникает трудность с правильным подбором типов линий и длиной штрихов. Это только дилетанту кажется, что нарисовано "пунктиром", для профессионалов важно точное соответствие стандарту. Рисование дорог будем выполнять путем трассировки одной стороны, указанием края другой стороны и выполнением эквидистантной линии (такой, какая выполняется командой OFFSET (ПОДОБИЕ)). Саму команду OFFSET (ПОДОБИЕ) мы применять не будем, это было бы слишком просто.

Основная функция (листинг 28.10) не сложна, но для ее реализации нам понадобятся несколько дополнительных библиотечных функций.

Листинг 28.10. Функция `ru-draw-road-topo-dline`

```
(defun ru-draw-road-topo-dline (msg ltype1 ltype2 /  last_pnt list_point
pre_last_pnt second_road_point vla_array_objs)
;| Примеры вызова:
(ru-draw-road-topo-dline "лесной дороги" "DASHED2" "DASHED2")
(ru-draw-road-topo-dline "грунтовой дороги" "CONTINUOUS" "DASHED")
|;
 (ru-app-begin)
 (if (setq list_point (reverse (_ru-trass-draw-lw (strcat "края " msg)
           nil nil nil T nil 0 0 nil)))
;| Нам потребуется угол между предпоследней и последней точками, и для облегчения
их извлечения делаем реверс списка точек нарисованной трассы
|;
  (progn
    (setq last_pnt (car list_point) pre_last_pnt (cadr list_point)
          second_road_point
         (ru-get-point-reguired (strcat "Второй край " msg) last_pnt))
    (if (ru-ltype-load ltype1)
;;; Вариант с изменением типа линии модификацией DXF-кода
        (ru-ent-mod (entlast) ltype1 6)
        (ru-msg-alert (strcat "\nНе найден тип линии" ltype1)))
;;;  Функция - заменитель команды OFFSET
    (if (setq vla_array_objs
          (ru-obj-ent-offset (entlast)
            ((if
              (ru-geom-is-point-right-by-axis
                second_road_point pre_last_pnt last_pnt)
                - +) (distance last_pnt second_road_point)))
   (if (ru-ltype-load ltype2)
;| А здесь применяем тип линии к массиву, т. к. при подобии могут образоваться
несколько объектов
|;
    (ru-obj-vla-array-mod vla_array_objs "Linetype" ltype2)
    (ru-msg-alert (strcat "\nНе найден тип линии" ltype2))))
   )
 )
```

```
 (ru-app-end)
 (princ)
)
```

Создание подобного объекта выполняется функцией, показанной в листинге 28.11. Функции передается VLA-объект или примитив и смещение создаваемой линии. При отрицательном значении смещения новые объекты создаются подобием влево, а при положительном смещении — вправо от направления основного примитива. Возвращается массив созданных объектов типа `Variant`, т. к. в некоторых случаях может быть создано несколько отдельных объектов.

Листинг 28.11. Функция `ru-obj-ent-offset`

```
(defun ru-obj-ent-offset (some dist)
                         ;|
при смещении отрицательном - влево, при положительном - вправо от направления линии
возвращает Variant (array of objects)
|;
 (ru-error-catch (function (lambda ()
  (cond
   ((= 'vla-object (type some)) (vla-offset some dist))
   ((= 'ename (type some))
     (vla-offset (vlax-ename->vla-object some) dist)))))
  (function (lambda (msg)
               (princ (strcat "\nRU-OBJ-ENT-OFFSET: " msg)) nil))
)
)
```

Для правильного задания знака смещения требуется функция, определяющая, вправо или влево от последнего сегмента трассы расположена указанная точка второго края (листинг 28.12).

Листинг 28.12. Функция `ru-geom-is-point-right-by-axis`

```
(defun ru-geom-is-point-right-by-axis (orient_pnt first_pnt second_pnt /
                                       right_first_pnt)
 (setq right_first_pnt
  (polar first_pnt (ru-geom-go-right (angle first_pnt second_pnt)) 1.0))
  (ru-geom-is-point-in-line right_first_pnt orient_pnt
    (inters first_pnt second_pnt orient_pnt right_first_pnt nil)
 )
)
```

Теперь мы можем включить в иллюстрированное XML-меню сколько угодно типов дорог, и не только дорог, но и других объектов, строящихся подобием от края. Нетрудно модифицировать приведенные функции и для построения объектов, подобных осевой линии трассы.

ГЛАВА 29

Несколько программ для сантехников

В этой главе мы рассмотрим несколько программ для облегчения работы специалистов, занимающихся разработкой различных систем, внутри которых что-то "течет" или движется — технологических трубопроводов, котельных, отопления, вентиляции, водопровода, канализации, газоснабжения, тепловых сетей.

При всем многообразии технических решений в этих разделах проекта изобразительная часть их стандартизирована. Большинство программ или функций, требующихся "сантехникам", мы уже рассмотрели. Если читатели-сантехники этого не заметили, то мы напомним.

Все, связанное с общими данными по рабочим чертежам (таблицы, типовые тексты), мы уже автоматизировали. Для рисования любых видов оборудования у нас имеется множество вариантов вставки блоков. Рисовать трассы и трубопроводы в одну линию мы также умеем — и стандартными командами, и с помощью наших программ. В линии, изображающие трубопроводы и воздуховоды, мы умеем врезать (арматуру, переходы) или привязывать (опоры, футляры) изображения, являющиеся блоками. В *главе 26* мы даже порассуждали о гидравлических и аэродинамических расчетах.

Осталось нам совсем немного — "мелочевка", наподобие надписей диаметров трубопроводов, рисования трубопроводов и воздуховодов "в три линии" и продольных профилей сетей.

Профили

Сразу закроем тему профилей — разрабатывать программы для их автоматизированного построения мы пока не будем. У нас имеются средства для профилей (рис. 29.1), но "для служебного пользования", а в систему ruCAD мы пока включаем только программу рисования заготовки профиля, это хоть небольшое, но облегчение.

Почему мы не делаем программы для настоящего построения профилей, ведь это так необходимо всем? Причин несколько.

Во-первых, если уж браться делать такую программу, то ее надо делать так, чтобы получить результат с ее помощью было бы легче, чем вручную. Мы видели несколько решений по разработке профилей и делать подобные не желаем, а наша пока не доведена до "товарного вида". Хорошая программа построения профилей должна:

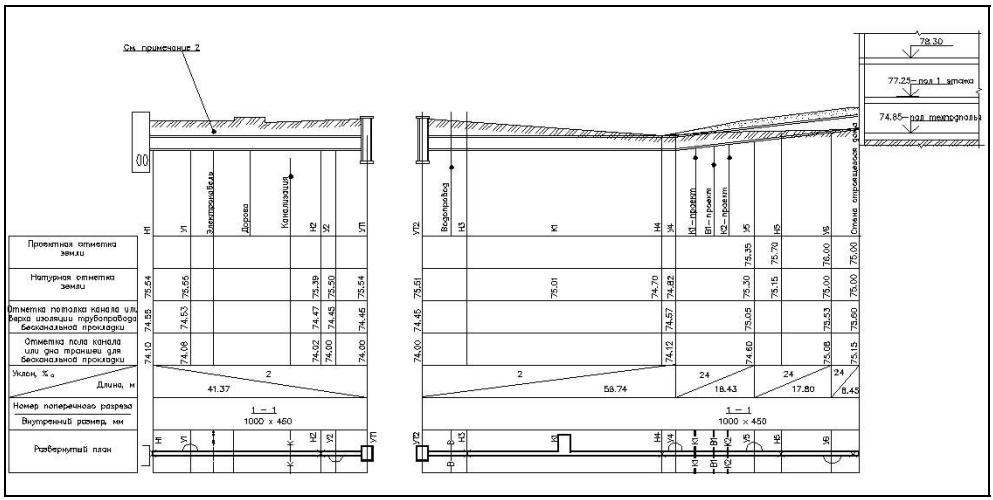


Рис. 29.1. Продольный профиль тепловых сетей

- получить от пользователя узловые точки трассы, указываемые на плане;
- автоматически определить "черные" отметки поверхности земли;
- автоматически или по запросу определить "красные" отметки земли;
- с помощью указания или автоматически определить пересечения с другими коммуникациями (плановое и вертикальное положение);
- построить заготовку профиля с грунтом и пересечениями;
- построить проектное положение трубопровода с учетом всех условий и ограничений (допустимые уклоны, глубина заложения, расстояния до пересекаемых коммуникаций).

Последний пункт является практически не поддающимся *полной* автоматизации в реальных условиях насыщенной городской застройки, но возможен интерактивный режим работы, позволяющий пользователю самому принимать решения. Все остальные во многом зависят от организации электронного плана города. Если у нас имеется электронный план *нашего* города, на котором все отметки, горизонтали и коммуникации находятся на стандартных слоях и выполнены стандартными средствами, то мы можем (это реализовано) автоматически определять отметки земли для узловых точек трассы. Но в *другом* городе эта программа будет бесполезной.

Во-вторых, понятие "всем надо" расплывчатое. Может быть надо и "всем", но оплачивать такую работу не хочет никто. Проектные организации, с которыми мы сотрудничаем, сейчас или не занимаются разработкой больших проектов наружных сетей, или делают это без электронной подосновы. В таких условиях им выгоднее разработать проект в "бумажном" виде. Например, подготовка электронного плана местности для разового проекта межпоселкового газопровода экономически нецелесообразна. Такой план проще сделать вручную, дополнив разработанными на компьютере общими данными и спецификациями. Небольшие участки трасс нетрудно "профилировать" на компьютере, но вручную.

Так что с профилями мы подождем до лучших времен.

Технология рисования "аксонометрии"

Больным вопросом для сантехников остается рисование *аксонометрических схем*. О проблемах, связанных с их рисованием, мы уже рассуждали в *главе 2*. Конечно, всем проектировщикам хочется "нажать на кнопку" и получить все и сразу, но не все понимают, что это невозможно.

Все сантехники хорошо знают, что такое "аксонометрия под 45 градусов", машиностроители и программисты с этим не сталкиваются, а знакомы с обычной изометрией. Во избежание недоразумений разберемся с терминологией.

"Схемы систем выполняют в аксонометрической фронтальной изометрической проекции в масштабе" — гласит пункт 3.2.1 ГОСТ 21.602-79. В учебной литературе по строительному черчению поясняют: "аксонометрические схемы выполняют во *фронтальной изометрии с левой системой осей и коэффициентом искажения вдоль осей, условно принятым за единицу*, что позволяет использовать метрический масштаб при построении". Если элементы схемы при вычерчивании в масштабе накладываются один на другой или непроизводительно занимают площадь чертежа, делают обрыв и выносят часть изображения на свободное место. Когда наложение элементов затрудняет чтение схемы, для увеличения наглядности прорезают линии, обнажающие трубопроводы, расположенные дальше от наблюдателя.

На рис. 29.2 показан куб с "трубопроводом", проведенным по трем ребрам, в различных аксонометрических проекциях.

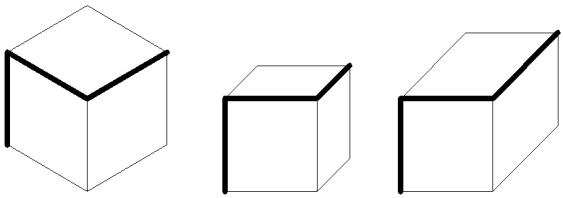


Рис. 29.2. Изображение куба в аксонометрических проекциях: *а* — изометрия, *б* — фронтальная диметрия с левой системой осей, *в* — фронтальная диметрия ("сантехническая аксонометрия")

Хуже всего, с "начертательно-геометрической" точки зрения, выглядит сантехническая аксонометрия (рис. 29.2, *в*), но именно она используется десятки лет по простой причине — для вычерчивания схемы достаточно было рейсшины и треугольника, размеры по всем осям откладывались просто по делениям без всяких вычислений. Измерить размеры было так же просто. Изометрия (рис. 29.2, *а*) неудобна для изображения длинных участков трубопроводов, т. к. площадь чертежа при этом используется нерационально.

Система AutoCAD имеет хорошие потенциальные возможности для трехмерного моделирования, в том числе трубопроводов. Потенциал постепенно реализуется в специализированных приложениях к AutoCAD.

Казалось бы, получить *изометрическое* изображение трубопровода автоматически просто — нужно нарисовать его в трехмерной модели, установить требуемый вид, в пространстве листа разместить *видовой экран* (Viewport) и показать в нем модель

трубопровода (рис. 29.3). Однако реалистичное изображение это не схема, на которой должны быть показаны с помощью специальных обозначений все элементы трубопровода, марки, диаметры, отметки, уклоны и т. п. Тонированное изометрическое изображение может быть прекрасным дополнением к рабочим чертежам, но не может заменить их. По трехмерной модели легко могут быть построены различные планы, фрагменты и виды. Сложнее, чтобы это были не просто виды, а чертежи видов, но и это уже решается. Появляются и программы, автоматически создающие изометрические изображения из трехмерной модели. При наличии всех требуемых проекций изометрические схемы вообще не нужны (в рабочем проекте, но могут понадобиться в монтажном проекте). При проектировании котельных, например, изометрические схемы традиционно не разрабатывались, но планы и разрезы дополнялись детальной плоской монтажной схемой трубопроводов.

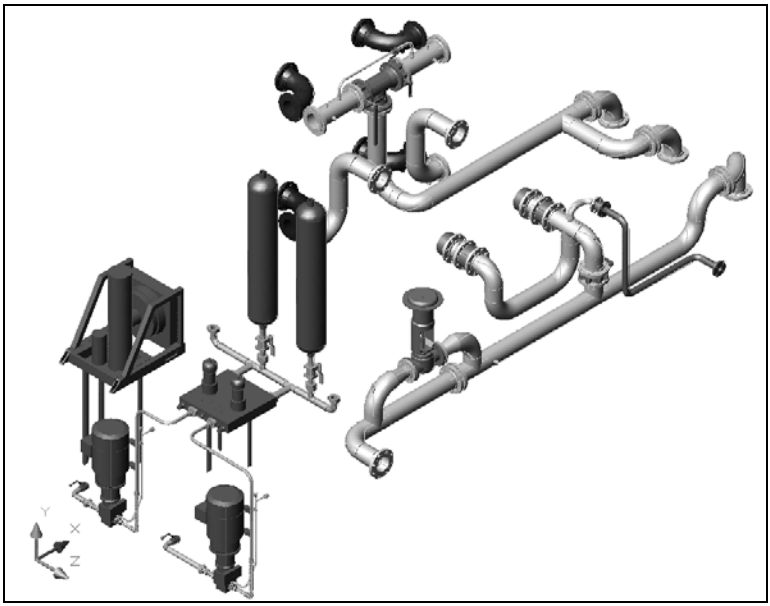


Рис. 29.3. Модель трубопроводов в изометрии (файл %Acad_2004%\Sample\Oil Module.dwg)

Однако чтобы обычный проектировщик-сантехник смог ощутить все преимущества моделирования трубопроводов, необходимо:

- иметь *точную* трехмерную строительную и технологическую подоснову всего здания или сооружения;
- иметь программные средства для моделирования трубопроводных и вентиляционных систем, адаптированные для использования российских изделий;
- уметь работать в трехмерной модели и прокладывать трубопроводы и воздуховоды в пространстве.

Многие ли проектные организации способны обеспечить выполнение этих условий? Конечно, такие фирмы есть, и их становится все больше. Если имеются заказы на проектирование объектов с большой стоимостью строительства и проектирования, т. е. возможность и приобрести программные средства, и обучить специалистов в

специализированных центрах подготовки, и разрабатывать документацию не на условиях "вчера надо было". Представляется, что соотношение количества таких фирм по отношению к обычным такое же, как соотношение граждан, ездящих на "Мерседесах", по отношению к остальному "несознательному" населению.

Модель здания и модель трубопроводов должны быть именно *абсолютно точными*, а не просто похожими на реальный объект. Точность необходима для того, чтобы использовать преимущества моделирования. Если строительная длина задвижки составляет 230 мм, то и в модели ее длина должна быть именно 230 единиц, а не "примерно 200—300". Трубопровод или воздуховод должен быть смоделирован со всеми отступами, "скобами", "утками" и прочими мельчайшими деталями. Сделать это для реального здания, особенно существующего, насыщенного оборудованием и коммуникациями вообще не всегда возможно.

Не случайно фирма Autodesk в качестве примера трехмерной модели приводит именно установку Oil Module (см. рис. 29.3). Это компактный блок оборудования, предназначенный для заводского изготовления. Преимущество трехмерной модели для проектирования (скорее, конструирования) *таких* объектов не вызывает сомнения. В этом случае, кроме сборки будут выпущены еще и чертежи узлов и деталей трубопроводов. Полный комплект займет целый альбом.

Недаром в комплект примеров системы AutoCAD не попал ни один комплексный *строительный проект* вместе с трехмерной сантехникой — он мог бы быть наглядным доказательством преимуществ моделирования.

Пример %Acad_2004%\Sample\8th floor hvac.dwg (посредственного, по российским меркам, чертежного качества) — всего лишь плоский чертеж, наложенный на трехмерную архитектурную модель.

Предположим, что вы все-таки построили качественную трехмерную модель трубопроводов. Теперь попробуйте сформировать на листах, предназначенных для печати, такие изометрические виды, по которым монтажники на стройке, имеющие дело только с бумажным чертежом, смогут изготовить узлы и смонтировать систему. Комплект чертежей *компактной* установки Oil Module можно подготовить без особых ухищрений, а с объектом, подобным 8th floor, не говоря уж о системах *длинного* производственного объекта, проблемы будут, и проблемы непростые. Например, трубопроводы малых диаметров и большой длины при параллельной прокладке будут просто сливаться в малопонятную линию, при этом мелкие изгибы будет трудно рассмотреть.

На следующем этапе попробуйте "пропустить"[1] такой проект через орган Госэкспертизы (об этих проблемах мы также писали в *главе 2*). Если проект выполнен иностранной фирмой и с нее нечего взять, или, наоборот, есть что взять и взяли, то экспертиза пройдет успешно. Но рядовая проектная фирма, в рядовом российском городе, по рядовому проекту запросто получит заключение о нарушении стандартов СПДС[2] и необходимости переработки проекта. Вероятность такого исхода обратно пропорциональна компетентности специалистов, проводящих экспертизу. Наблюдается устойчивая тенденция "цепляния" экспертов за мелкие детали — "передвинуть

[1] Вообще-то получение заключения экспертизы является обязанностью заказчика (застройщика), но без участия проектировщиков это мероприятие не обходится.

[2] Система проектной документации для строительства.

стояк в другой угол", "подключиться к другому колодцу", "длину кожуха предусмотреть кратной 4-м метрам". Настоящие ошибки, например неверный расчет воздухообмена во влажном помещении, приводящий к намоканию строительных конструкций, могут быть не замечены, а "выполнение аксонометрических схем в проекции, не соответствующей стандартам СПДС", отметят непременно.

Отсталость стандартов от современных компьютерных технологий — отдельная тема. Изменять стандарты нужно, но вряд ли это будет сделано скоро. Пока выход в том, что их следует придерживаться, постепенно формируя общественное мнение о необходимости перехода на новые технологии. Практически это означает, что, даже имея модель трубопроводов, следует, не тратя времени на бесполезные споры, просто уметь быстро чертить на плоскости традиционные аксонометрические схемы, показывая на них то, что не отражено в модели.

Одна из задач нашей системы — предоставить инструменты для быстрого рисования таких схем.

Техника рисования схем проста — или сначала нарисовать трубопроводы полилиниями, а затем врезать в них, с помощью наших программ, условные обозначения элементов трубопроводов, или сначала нарисовать крупные элементы и узлы, соединить их линиями, а затем врезать в линии "мелочевку". Для удобного проведения линий под требуемыми углами в современных версиях системы AutoCAD удобнее всего использовать режим **POLAR** с установкой шага углов 45 градусов (диалоговое окно **Drafting Settings** (Режимы рисования), вкладка **Polar Tracking** (Отслеживание)). Чтобы пользователям не приходилось обращаться к этим средствам, мы просто включим в настройки нашей системы установки соответствующих системных переменных, а переключение режима **POLAR** всегда доступно в строке статуса.

Замечание

Если понадобится рисование в классической изометрии (рис. 29.2, *а*), то ее можно включить на вкладке **Snap and Grid** (Шаг и сетка) окна **Drafting Settings** (Режимы рисования) переключателем **Isometric snap** (Изометрическая), а переключение между "плоскостями" изометрии во время работы делать нажатием клавиши <F5>.

Средства рисования аксонометрии в системе ruCAD

Фактически у нас сделано почти все. Рисовать трубопроводы в схеме можно обычными полилиниями, а потом только врезать в них арматуру и другие элементы. Однако есть некоторые нюансы, связанные с шириной линий. Подавляющее большинство линий, изображающих трубопроводы и воздуховоды, должны быть нарисованы основной линией. Отдельные участки трубопроводов, а также арматура и другие изделия должны быть нарисованы тонкими линиями. В соответствии с концепцией нашей системы для рисования "жирных" линий используется *вес* (lineweight) *линий*. О достоинствах и недостатках веса линий мы рассуждали в *главе 3*, но решили, что достоинств больше, чем недостатков. В отношении линий в схемах это не так.

Все схемы насыщены врезками изображений элементов, выполненными тонкими линиями, и важно, чтобы торцы разрезанной линии были прямоугольными и не "влезали" в изображение элемента трубопровода.

На рис. 29.4 слева показан фрагмент схемы, в котором трубопровод нарисован полилинией с заданной шириной, а справа — полилинией с нулевой шириной, но с

заданным весом. Отчетливо видна небрежность правого рисунка. Кому-то "и так сойдет", а кому-то эти мелкие неточности неприемлемы.

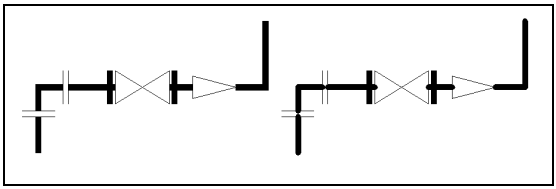


Рис. 29.4. Фрагмент схемы. Слева — задана ширина полилиний, справа — задан вес линий

При уменьшении масштаба изображения на экране (рис. 29.5) недостаток веса линий усугубляется. На отпечатанном листе бумаги этот недостаток может быть незаметен, а проявится достоинство веса линии — постоянная ширина на бумаге независимо от масштаба печати, но очень часто более важным является именно качественное экранное изображение. Схемы насыщены *внемасштабными символами* и чертежи схем должны печататься не в произвольном, а в заданном при рисовании масштабе печати. В этом случае достоинство постоянности ширины линий, использующих вес, не выявится, а недостатки останутся.

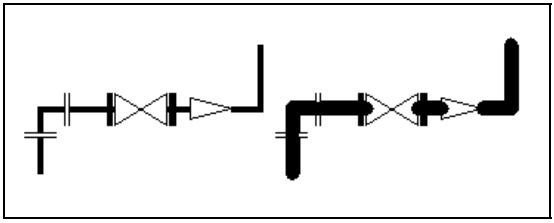


Рис. 29.5. Фрагмент схемы в уменьшенном масштабе изображения на экране

Вообще-то в **Plot Style Table Editor** (Редактор таблиц стилей печати) (рис. 29.6) на вкладке **Table View** (Таблица) имеется настройка **Line End Style** (Стиль окончания линий), позволяющая установить стиль печати концов линий ("Butt", "Square", "Round" и "Diamond"). От круглых торцов на печати можно избавиться или, наоборот, сделать их "бриллиантовыми", но это только при выводе на бумагу. Да и много ли рядовых пользователей знают про такие, столь далеко запрятанные детали настройки?

Замечание

В настройке сопряжения линий **Line Join Style** (Стиль соединения линий) имеется опция **Use object join style** (Взять стиль соединения из объекта), но объектов, имеющих такой параметр, нет. В том числе в самой последней версии AutoCAD 2005.

В результате, после консультаций с пользователями наших прежних систем мы приняли решение — для рисования схем использовать старый надежный метод рисования полилиниями с задаваемой шириной. Пока только для схем, хотя пользователи просят оставить ширину и для деталировок трубопроводов и воздуховодов.

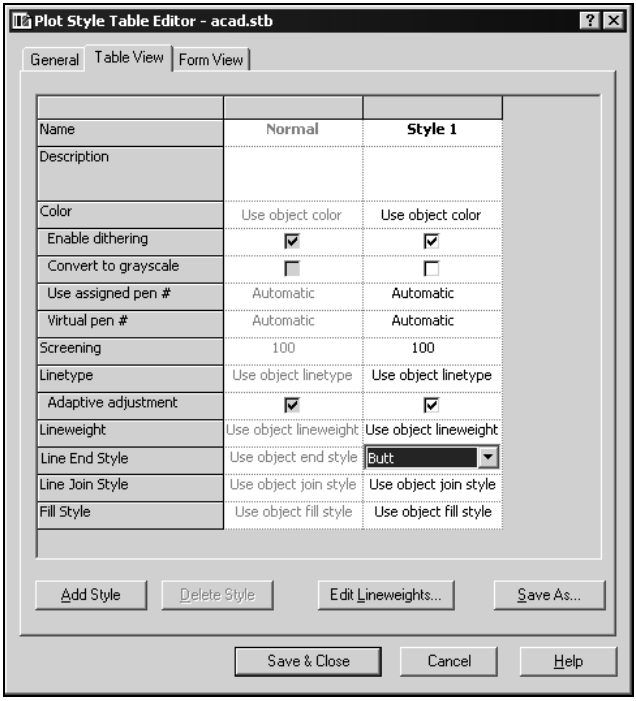


Рис. 29.6. Настройка торцов линий

Узлы схем

Кроме стандартных *внемасштабных* условных обозначений элементов схем (арматура, переходы, соединения деталей) удобно использовать блоки укрупненных типовых узлов. Такие блоки, как и любые реальные изделия, следует рисовать в натуральную величину, принимая за единицу рисунка миллиметр. Однако в подобных блоках обязательно будет иметься схематичное изображение трубопровода, а трубы в чертежах разных масштабов могут иметь разную ширину. Способ изменения ширины полилиний в блоке был рассмотрен в *главе 22*. В блоках, для которых потребуется изменение ширины линий, линии, у которых ширина будет меняться, должны быть нарисованы компактными полилиниями (LWPOLYLINE), а остальные — любыми другими примитивами. На рис. 29.7 показаны блоки типовых радиаторных узлов систем отопления. Блоки предназначены для вставки в общую схему системы, где излишняя детализация не требуется и даже вредна, поэтому в узлах с замыкающими участками не показаны регулирующие краны.

Деталировка узлов обычно производится в более крупном масштабе и для нее предназначены другие блоки (рис. 29.8).

Впрочем, нет препятствий для вставки блоков с кранами и в общую схему, но при масштабе печати 1:100 изображения кранов будут слишком мелкие, линии сольются и вместо кранов будут напечатаны непонятные "козявки".

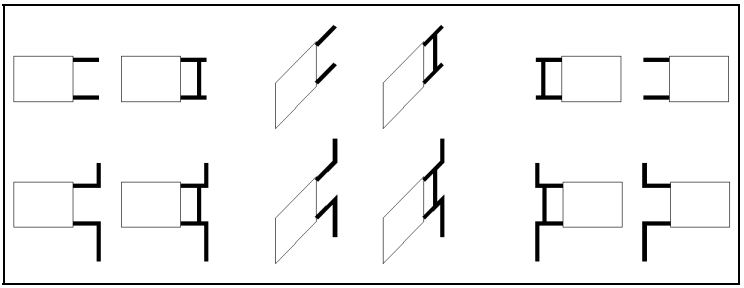


Рис. 29.7. Упрощенные схемы типовых радиаторных узлов систем отопления

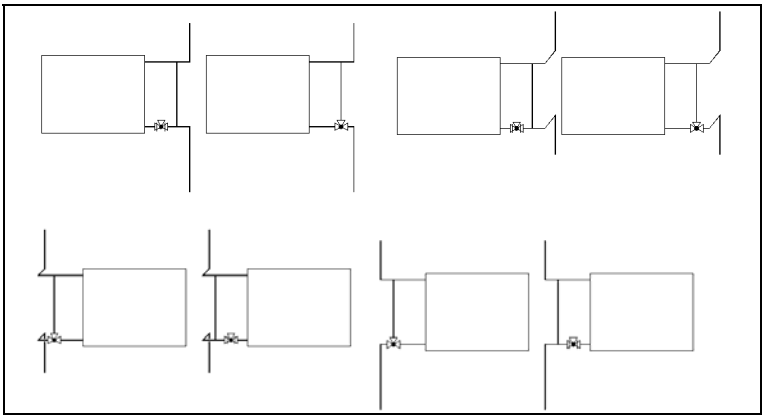


Рис. 29.8. Типовые радиаторные узлы с регулирующими кранами

Замечание

Обратите внимание, в схемы узлов не включены никакие внемасштабные символы, например, выноски, знаки диаметров. Это позволяет использовать блоки узлов в чертежах, предназначенных для печати в любом масштабе.

Вставка библиотечных блоков с изменяемой шириной выполняется рассмотренной ранее функцией `ru-block-lw-lib-insert-ptask-angleask`. Блоки для схем обычно вставляются с поворотом 0 градусов, поэтому нам потребуется еще один вариант функции (листинг 29.1). Для этого варианта нам пришлось немного усовершенствовать функцию `ru-block-lw-insert-ptask-angleask`, введя в нее дополнительный параметр — угол поворота (если задан `nil`, то угол поворота запрашивается при вставке).

Листинг 29.1. Функция `ru-block-lw-lib-insert-ptask`

```
(defun ru-block-lw-lib-insert-ptask (block_lib  block_name scale)
 (ru-app-begin)
 (if (ru-block-lib-insert block_lib block_name)
   (ru-block-lw-insert-ptask-angleask block_name scale 0))
   (ru-app-end) (princ))
```

Теперь мы можем написать XML-меню радиаторных узлов (листинг 29.2).

Листинг 29.2. Фрагмент XML-меню радиаторных узлов

```
<item name='Узлы радиаторные' macro=''>
  <item name='Проточные' macro=''>
   <item name='Без стояков' macro=''>
    <item name='Левый' image='OV\heat_rad_001.GIF'
      comment='Радиаторный узел проточный левый'
      macro='(ru-block-lw-lib-insert-ptask "ru-lib-heater-lw"
"РАДИАТОР_ПРОТ_Л_ФРОНТ" 1)'/>
    <item name='Правый' image='OV\heat_rad_002.GIF'
      comment='Радиаторный узел проточный правый'
      macro='(ru-block-lw-lib-insert-ptask "ru-lib-heater-lw"
"РАДИАТОР_ПРОТ_ПР_ФРОНТ" 1)'/>
   </item>
  </item>
 </item>
</item>
```

Вид XML-меню радиаторных узлов во время выбора показан на рис. 29.9.

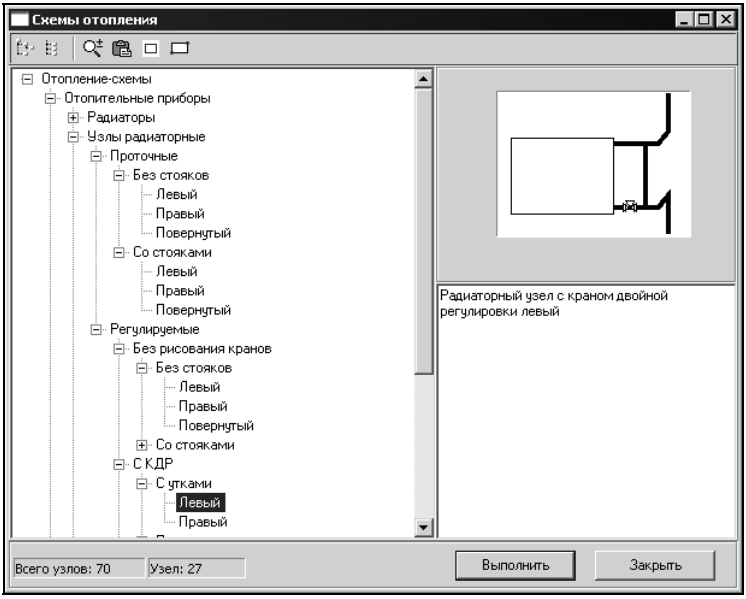


Рис. 29.9. XML-меню радиаторных узлов

Рисование линий схем

Рисовать линии схем мы договорились обычными полилиниями. Для облегчения установки ширины полилиний мы разработаем дополнительную функцию, пол-

ностью имитирующую работу команды PLINE (ПЛИНИЯ), но начинающую рисование с требуемой шириной (листинг 29.3).

Листинг 29.3. Функция ru-pline-draw-with-width

```
(defun ru-pline-draw-with-width (pline_width)
 (ru-error-save-sysvars (list (list "CMDECHO" 1) (list "BLIPMODE" 0)
  (list "OSMODE") (list "PICKBOX") (list "HIGHLIGHT") (list "ORTHOMODE")
  (list "PLINEWID"  pline_width)))
  (vl-cmdf "_.PLINE")(while (> (getvar "CMDACTIVE") 0) (vl-cmdf pause))
  (ru-error-restore-sysvars) (princ))
```

Эта функция будет применяться очень часто и достойна не только включения в обычное меню (листинг 29.4), но и выделения кнопок на панели инструментов. Как правило, используются тонкая или основная линия.

Листинг 29.4. Макросы меню рисования типовых полилиний

```
[Тонкая полилиния]^C^C^P(defun C:RU()(ru-pline-draw-with-width 0));RU
[Основная полилиния]^C^C^P(defun C:RU()(ru-pline-draw-with-width
(ru-normal-pline-width)));RU
```

Пересечение линий

При вычерчивании схем часто для увеличения наглядности прорезают линии, обнажающие трубопроводы, расположенные дальше от наблюдателя. Делают это обычно командой BREAK (РАЗОРВАТЬ), но и эту процедуру можно облегчить (листинг 29.5). Программа просит указать нижнюю и верхнюю линии, определяет ширину пересечения с учетом физической ширины или веса верхней линии и делает требуемый разрыв.

Листинг 29.5. Файл ru_cross_lines.lsp

```
(defun START (/angle_segm   break_length can_break ent1 ent2 ent_point1 ent_point2
inters_pnt lst_segm1 lst_segm2 pnt1 pnt1_break pnt1_found pnt2 pnt2_break
pnt2_found pnt3 pnt4 _width)
 (defun _width (ent / obj w)
 (setq obj (vlax-ename->vla-object ent)
    w (ru-conv-millimeter-in-paper-to-unit 0.5))
  (max (+ (vla-get-lineweight obj) w)
     (if (= (vla-get-objectname obj) "AcDbPolyline")
                 (+ (vla-get-constantwidth obj) w) 0) w))
;;------------- Основная функция ------------------------------
 (while (setq ent_point1
   (ru-get-point-on-ent "Укажи точку на разрываемом сегменте"))
  (if (setq lst_segm1 (ru-geom-list-ent-point
    (setq ent1 (car ent_point1)) (cadr ent_point1)))
   (progn
     (setq pnt1 (car lst_segm1) pnt2 (cadr lst_segm1) can_break t)
```

```
    (if (setq ent_point2
       (ru-get-point-on-ent "Укажи точку на верхнем сегменте"))
       (if (setq lst_segm2 (ru-geom-list-ent-point
                        (setq ent2 (car ent_point2)) (cadr ent_point2)))
        (setq pnt3 (car lst_segm2) pnt4 (cadr lst_segm2) can_break T)
        (setq can_break nil)))
  )
   (setq can_break nil)
 )
  (if (and can_break (setq inters_pnt (inters pnt1 pnt2 pnt3 pnt4 T)))
    (progn
     (setq angle_segm (angle pnt1 pnt2)
          break_length (* (_width ent2) 0.5)
          pnt1_break (polar inters_pnt angle_segm break_length)
          pnt2_break (polar inters_pnt (ru-geom-go-back angle_segm)
                                          break_length))
    (if (>= (distance inters_pnt pnt1_break) break_length)
       (setq pnt1_found T)
       (setq pnt1_found nil
            pnt1_break (polar inters_pnt angle_segm (* break_length 2))))
    (if (>= (distance inters_pnt pnt2_break) break_length)
       (setq pnt2_found T)
       (setq pnt2_found nil
            pnt2_break (polar inters_pnt (ru-geom-go-back angle_segm)
                                              (* break_length 2))))
    (ru-var-clear-osnap)
   (cond
    (pnt1_found
      (command "_.BREAK" (list ent1 (trans inters_pnt 0 1)) "_F"
                       (trans pnt1_break 0 1) (trans pnt2_break 0 1)))
    (pnt2_found
      (command "_.BREAK" (list ent1 (trans inters_pnt 0 1)) "_F"
           (trans pnt2_break 0 1) (trans pnt1_break 0 1)))
    (T (ru-msg-alert "\nНе могу найти пересечение... "))
  ) (ru-var-restore-osnap)))) (princ)
)
(START)
```

Выноски диаметров

В сантехнических чертежах проставляется несметное количество подписей диаметров трубопроводов, изделий и воздуховодов. Диаметры приходится рисовать разными способами, и мы приведем несколько примеров программ, облегчающих эту работу. Сначала напишем программу для рисования диаметра вдоль линии. Функции для рисования текста вдоль линий мы рассматривали в *главе 22*. Подпись диаметра отличается от обычного текста наличием знака диаметра. Проще всего добавлять *знак диаметра* с помощью специального символа `%%c`, но в этом случае имеется вероятность, что используемый шрифт не поддерживает такие специальные символы. Теперь уже трудно и найти "плохие" шрифты, но ведь где-то находят, и чертежи могут потерять первоначальный вид. Поэтому мы до сих пор знак диаметра заменя-

ем специальным блоком, который сохранит неизменный вид. Программа для рисования диаметров вдоль линий приведена в листинге 29.6.

Листинг 29.6. Файл ru_draw_dia_along_line.lsp

```
(defun START (/ diameter _dia-up-line)
  (defun _dia-up-line (txt / lst_break result ang_rad)
   (if (setq lst_break (ru-pline-break-length
         (strcat "Точка на линии для диаметра '" txt "'") 0))
   (progn
     (setq ang_rad (cadr lst_break))
     (if (and (> ang_rad 1.5708) (<= ang_rad 4.71239))
           (setq ang_rad (ru-geom-go-back ang_rad)))
     (ru-draw-diameter-text (car lst_break) txt ang_rad)
     (setq result t))) result)
;;;-------- Основная функция ----------------------------------------
 (ru-app-begin)
 (if (setq diameter (ru-get-diam-text
               (ru-user-read-last-param "LastDiameterString" "100")))
  (progn
      (while (_dia-up-line diameter))
      (ru-user-write-last-param "LastDiameterString" diameter)))
  (ru-app-end) (princ)
)
(START)
```

Функция непосредственного рисования диаметра приведена в листинге 29.7. Она возвращает набор из блока и текста диаметра для последующих возможных манипуляций.

Листинг 29.7. Функция ru-draw-diameter-text

```
(defun ru-draw-diameter-text (pnt text ang_rad / ent_last)
  (setq ent_last (entlast))
  (ru-text-add text
    (ru-draw-block-diameter pnt (ru-normal-text-height) ang_rad)
    (ru-normal-text-height) ang_rad acalignmentleft)
  (ru-ss-select-after-ent ent_last)
)
```

Вставка блока знака диаметра осуществляется функцией, приведенной в листинге 29.8 и возвращающей точку, от которой можно продолжать текст значения диаметра. Для маловероятного случая отсутствия библиотечного блока предусмотрено символьное написание знака диаметра.

Листинг 29.8. Функция ru-draw-block-diameter

```
(defun ru-draw-block-diameter
       (ins_pnt text_height ang / block_name blk_pnt txt_pnt)
 (setq txt_pnt ins_pnt block_name "ru_diameter")
```

```
 (if (ru-block-lib-insert "ru-lib-common" block_name)
  (progn
    (ru-block-insert-obj block_name
        (polar (polar ins_pnt (ru-geom-go-left ang) text_height)
           ang (* text_height 0.5)) text_height text_height 1 ang)
      (polar (polar ins_pnt (ru-geom-go-left ang) (* text_height 0.5))
             ang (* text_height 1.5)))
  (progn
    (ru-text-add "D=" (setq ins_pnt (polar ins_pnt (ru-geom-go-left ang)
          (* text_height 0.5))) text_height ang acAlignmentLeft)
    (polar ins_pnt text_height (* text_height 2.0))))
)
```

Текст диаметра задается в диалоговом окне, позволяющем ввести строку вручную, взять указанием надписи в рисунке или выбрать из XML-меню диаметров различных изделий (листинг 29.9).

Листинг 29.9. Функция `ru-get-diam-text`

```
(defun ru-get-diam-text (default / lst)
  (if (setq lst (ru-dlg-get-two-string "Ввод диаметра" "Диаметр"
      default T "" "" T nil "options\\dia_lst.xml" ""))
    (car lst) nil))
```

Теперь напишем программу рисования диаметров с полочкой выноски. Программа (листинг 29.10) позволяет выбрать диаметр из справочника, указать несколько изделий с одинаковым диаметром. Для повышения динамичности работы использован аргумент, позволяющий под полочкой выноски диаметра рисовать дополнительный текст (марку изделия, обозначение стояка и т. п.), вводимый вручную или выбираемый из XML-меню.

Листинг 29.10. Файл ru_draw_leader_dia_with_text.lsp

```
(defun START (draw_mark / align ang_txt_line diameter dia_mark end_leader
end_txt_line len_txt lst lst_txt start_leader start_txt start_txt_line)
  (defun _move (selection pnt)
    (while (not (ru-error-catch (function (lambda ()
              (princ "\nТочка вставки марки изделия: ")
              (vl-cmdf "_.MOVE" selection "" (trans pnt 0 1) pause)))
          nil))))
;;; ------------------------ Основная функция ------------------------
 (ru-app-begin)
 (while
  (setq diameter (ru-get-diam-text
        (ru-user-read-last-param "LastDiameterString" diameter)))
  (if (setq start_leader (ru-get-point-or-exit "Начало выноски" nil))
   (progn
    (setq end_leader (ru-get-point-reguired "Конец выноски" start_leader)
        align (ru-text-end-leader-align (angle start_leader end_leader))
       lst_txt (textbox (list (cons 1 diameter)))
```

```
      len_txt (+ (- (car (cadr lst_txt)) (caar lst_txt))
                          (* (ru-normal-text-height) 2.5))
      ang_txt_line (if (= align acalignmentleft) 0 (ru-geom-go-back 0))
      start_txt_line (if (= align acalignmentleft)
              end_leader (polar end_leader (ru-geom-go-back 0) len_txt))
      end_txt_line (polar end_leader ang_txt_line len_txt)
      start_txt (polar start_txt_line (ru-geom-go-left 0)
                               (* 0.1 (ru-normal-text-height))))
   (ru-draw-diameter-text start_txt diameter 0)
   (ru-pline-add (list start_leader end_leader end_txt_line) nil 0 0 nil)
   (while (setq start_leader
           (ru-get-point-or-exit-base "Следующее изделие" end_leader))
          (ru-line-add start_leader end_leader 0 nil))
;;; ------- Вариант рисования дополнительного текста под полочкой
   (if draw_mark (progn
    (if (setq lst (ru-dlg-get-two-string "Продолжение текста диаметра"
         "Марка"
         (setq dia_mark (ru-user-read-last-param "LastDiameterMark"
         "Ст В1.1")) T "" "" T nil "options\\dia_txt.xml" ""))
      (progn
        (ru-text-add (setq dia_mark (car lst))
          (setq start_txt (polar start_txt (ru-geom-go-right 0)
           (* 1.2 (ru-normal-text-height))))
          (ru-normal-text-height) "0" nil)
;;; Даем возможность передвинуть текст
       (_move (entlast) start_txt)))
   (ru-user-write-last-param "LastDiameterMark" dia_mark)))))
 (ru-user-write-last-param "LastDiameterString" diameter)
 ) (ru-app-end)(princ))
```

Пример использования приведенных программ показан на рис. 29.10.

Рисование элементов трубопроводов в три линии

Изображение трубопроводов диаметром более 100 мм "в три линии[1]" предписывается стандартами для фрагментов и узлов (в стандарте на чертежи котельных — трубопроводов, "у которых на чертеже диаметры равны 2 мм и более", что правильнее).

Такое изображение требуется не только "для красоты", но и для того, чтобы правильно сконструировать трубопровод с учетом размеров фасонных частей, арматуры и других изделий. Традиционно высока культура рисования трубопроводов у проектировщиков технологической части котельных — там очень много узлов в стесненном пространстве, требующих тщательной проработки, не выполняются аксонометрические схемы, способствующие "прокладке по соображению". При проектировании котельных всегда прорисовывались тройники, отводы и другие фасонные части.

[1] В стандартах пишут "двумя линиями", проектировщики говорят про три — есть еще и осевая линия трубопровода.

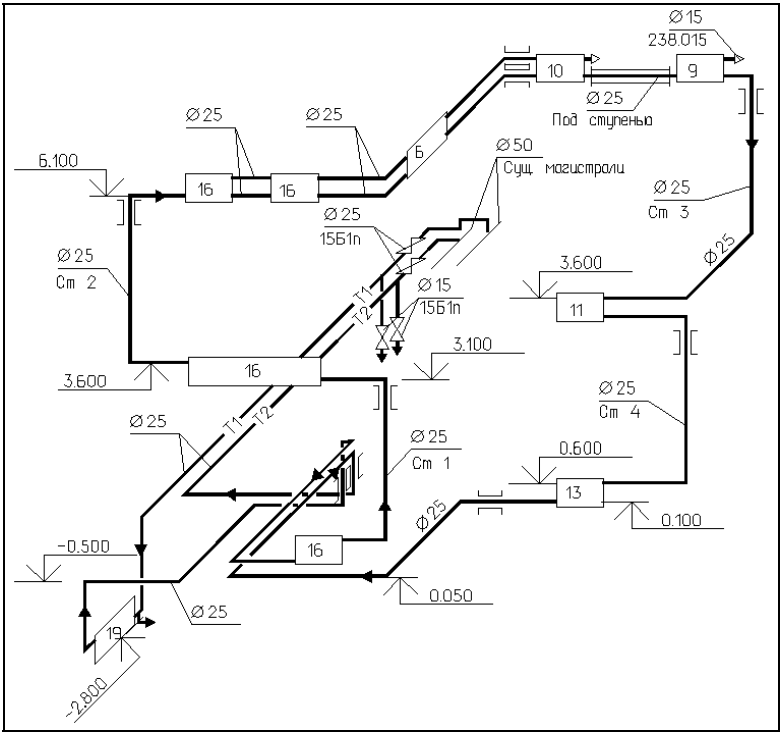


Рис. 29.10. Схема системы отопления

В ГОСТ 21.606-95 по выполнению рабочей документации тепломеханических решений котельных эта хорошая привычка узаконена, а вот в ГОСТ 21.601-79 на рабочие чертежи водопровода и канализации, на черт. 10, с примером плана насосной детали не проработаны, и это приводит к неверным техническим решениям (сварные стыки в толще стены, например). В *главе 3* мы писали о возможности избыточной детализации чертежей, а теперь приведем конкретный пример. На рис. 29.11 показана та же насосная станция, которая изображена и на черт. 10 в ГОСТ 21.601-79.

Наш чертеж содержит избыточную, по сравнению со стандартом, детализацию. Стандартизация строительного черчения, кроме достижения единообразия оформления, преследовала цели упрощения и ускорения работ. Разрабатывать вручную чертежи масштаба 1:50 так детально, как показано на рис. 29.11, будут только отдельные энтузиасты — слишком долго, "за свои харчи" и все равно только приблизительно точно. На компьютере этот рисунок мы, в присутствии "понятых", нарисовали за 27 минут. Все, что на нем показано, нарисовано с помощью программ системы ruCAD, за исключением стен и размеров, для которых использовались стандартные команды системы AutoCAD. Читатели могут сравнить впечатления от стандартного чертежа и нашего.

Но, кроме необязательной "красивости", детальный чертеж имеет более важные свойства. На рис. 29.12 показан увеличенный фрагмент рисунка в месте подключения маленького водопроводного насоса. Видно, что на деталях проработаны даже

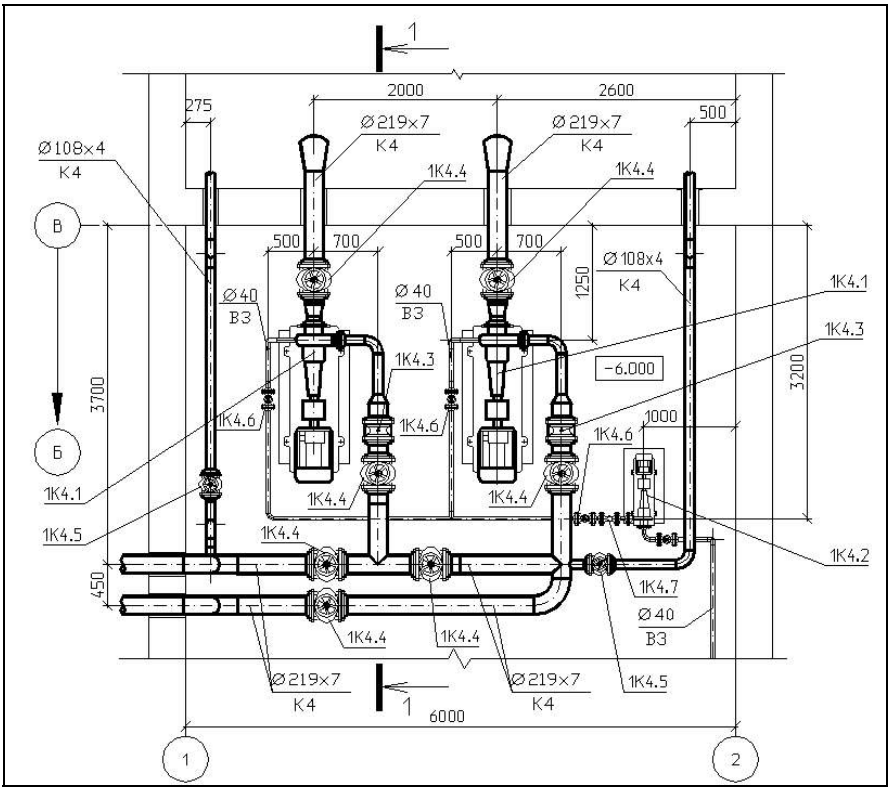


Рис. 29.11. План насосной

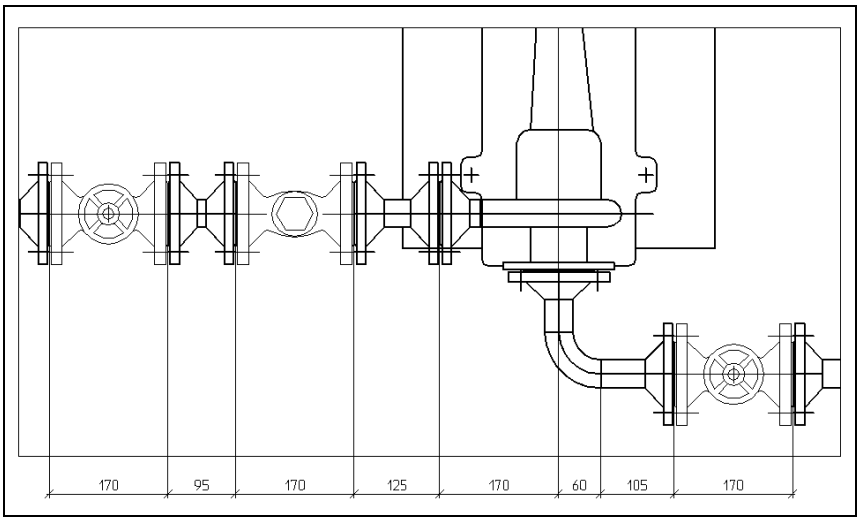


Рис. 29.12. Фрагмент плана насосной

выступы фланцев, т. е. чертеж выполнен очень точно. Это позволяет использовать его не только как *рабочий* чертеж, но и как *деталировочный* для монтажно-заготовительного производства.

Совмещение рабочего и монтажного проектирования, только начинавшееся в СССР, теперь легко может быть реализовано, особенно при росте количества проектно-строительных организаций, у которых конечным продуктом являются не рабочие проекты, а объекты, "сдаваемые под ключ".

В отличие от систем трехмерного моделирования, которым, безусловно, принадлежит будущее, двумерные системы с их "доступной плоской формой" легко осваиваются "тетками", а разрабатывать их может любой программист. Как это делается, мы рассмотрим ниже.

Программирование рисования трубопроводов с детализацией

Все трубопроводы и их элементы, показанные на рис. 29.11, могут быть нарисованы при выборе одного пункта меню — **СпецРис | Трубопроводы | Детализация**. При этом вызывается XML-меню, показанное на рис. 29.13.

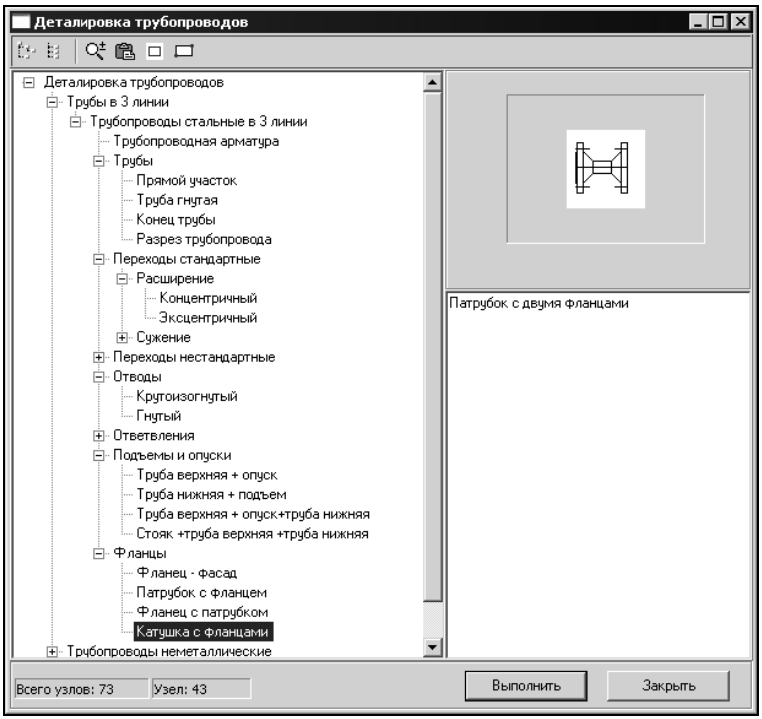


Рис. 29.13. XML-меню деталировки трубопроводов

Это XML-меню имеет несколько десятков элементов, в каждом из которых спрятан макрос, загружающий небольшую программу рисования конкретного элемента трубопровода (на рисунке выбран патрубок с двумя фланцами). Фрагмент XML-меню приведен в листинге 29.11.

Листинг 29.11. Фрагменты файла ru_pipe_detail.ruxm

```
<?xml version='1.0' encoding='windows-1251'?>
<root name='Трубопроводы стальные в 3 линии' macro=''>
 <item name='Трубопроводная арматура'
  image='TRUBA\ARMREAL\ARM_ALL.GIF'
  comment='Реалистичное изображение выбранного изделия'
  macro='(progn (ru-app-load "ru_san_pipe_detail")(START "Арматура"
(quote ru-pipe-valve-details) "Начало изделия со стороны входа среды"
"Направление потока" NIL "valve"))'/>
  <item name='Трубы' macro=''>
   <item name='Прямой участок' image='TRUBA\PIPES\PIPE.GIF'
    comment='Реалистичное изображение выбранного изделия'
    macro='(progn (ru-app-load "ru_san_pipe_detail")(START "Прямой участок" (quote
ru-pipe-line) "Начало участка трубы" "Конечная точка участка трубы" NIL
"pipe_line"))'/>
    <item name='Труба гнутая' image='TRUBA\PIPES\PIPE_GIB.GIF'
     comment='Реалистичное изображение выбранного изделия'
     macro='(progn (ru-app-load "ru_san_pipe_detail")(START "Труба гнутая" (quote
ru-pipe-line-bend) "Начало гнутой трубы" "Вторая точка гнутой трубы" NIL
"pipe_bend"))'/>
    <item name='Конец трубы' image='TRUBA\PIPES\PIPE_END.GIF'
    comment='Реалистичное изображение выбранного изделия'
    macro='(ru-app-load "ru_san_pipe_end")'/>
    <item name='Разрез трубопровода' image='TRUBA\PIPES\PIPE_RAZ.GIF'
    comment='Разрезы труб с опорами и без опор'
     macro='(ru-app-load "ru_san_pipe_sectional")'/>
  </item>
  <item name='Фланцы' macro=''>
    <item name='Фланец - фасад' image='TRUBA\PIPES\FLAN_FA.GIF' comment='Фланец с
фасада' macro='(progn (ru-app-load "ru_san_pipe_flange")(START "Фланец с фасада"
(quote ru-pipe-flange-face)))'/>
    <item name='Патрубок с фланцем' image='TRUBA\PIPES\FLAN_PF.GIF'
comment='Рисование от патрубка' macro='(progn (ru-app-load
"ru_san_pipe_flange")(START "Патрубок с фланцем" (quote ru-pipe-branch-flange)))'/>
    <item name='Фланец с патрубком' image='TRUBA\PIPES\FLAN_FP.GIF'
comment='Рисование от фланца' macro='(progn (ru-app-load
"ru_san_pipe_flange")(START "Фланец с патрубком" (quote ru-pipe-flange-branch)))'/>
    <item name='Катушка с фланцами' image='TRUBA\PIPES\FLAN_FF.GIF'
comment='Патрубок с двумя фланцами' macro='(progn (ru-app-load
"ru_san_pipe_flange")(START "Катушка с фланцами" (quote ru-pipe-flange-flange)))'/>
  </item>
</root>
```

При просмотре полного текста меню на компакт-диске читатели убедятся, что почти все рисование сосредоточено в одном файле ru_san_pipe_detail.lsp. Кроме того, используются ru_san_pipe_sectional.lsp и ru_san_pipe_flange.lsp. Две последних програм-

мы предназначены для рисования специфичных изображений, и мы о них поговорим потом.

Программа ru_san_pipe_detail рисует прямые участки, гнутые трубопроводы, переходы, отводы, тройники, врезки, подъемы и спуски. Все эти узлы обладают общими свойствами:

- каждый узел имеет главный параметр — диаметр основного трубопровода и может иметь дополнительный параметр — диаметр ответвления или второй диаметр перехода;
- некоторые однотипные рисунки отличаются характером изображения (концентричный или эксцентричный переход, центральный угол отвода, дополнительные изображения на стояках);
- трубопроводная арматура имеет множество конструкций (задвижки, клапаны и т. п.), отличается давлениями, размерами, характером изображения (вид сбоку, сверху, вдоль оси), но также имеет главный параметр — диаметр;
- все элементы могут создаваться с разным весом линии.

Для каждого элемента необходимо задавать требуемые параметры, причем непременно в диалоговом окне. В системе AutoCAD R10 диалоговые окна отсутствовали, параметры отображались и изменялись с помощью опций командной строки и экранного меню. Это было не очень удобно, и с появлением диалоговых окон работа значительно упростилась.

Очень важно, что отдельные элементы нужны не сами по себе, а должны формировать связанное изображение трубопровода. На рис. 29.11 видно, что, начиная от напорного патрубка правого насоса, последовательно нарисованы: фланец, прямой участок, отвод, прямой участок, переход, патрубок с фланцем, обратный клапан, "катушка", задвижка, фланец, прямой участок, отвод, прямой участок, фланец, задвижка, фланец, прямой участок, подъем с двумя отводами в вертикальной плоскости, прямой участок и обрыв трубопровода. В этот трубопровод врезаны еще два трубопровода, состоящих из собственных цепочек элементов.

При выборе для рисования любого элемента, пользователь должен иметь возможность начать его рисование от введенной точки или продолжить рисование от предыдущего элемента, не затрачивая усилий на прицеливание.

В *главе 28* мы разбирали применение для рисования трасс функции `ru-trass-draw-first-prev-sample` *(см. главу 22)*, с помощью которой возможно рисование любых трасс с зацепкой за окончание предыдущего сегмента трассы. Подобную функцию можно разработать и для трубопроводов. При разработке таких программ возможны различные принципиальные подходы.

Во-первых, можно разработать большую программу с большим диалоговым окном, позволяющим выбрать элемент, установить его параметры и нарисовать. При таком подходе множество проблем имеет программист — требуется увязать множество условий и событий. Разработчик будет обречен всю жизнь совершенствовать эту программу, а пользователям сложные диалоги со множеством визуальных элементов неудобны.

Во-вторых, можно вынести выбор элементов в XML-меню, для групп однотипных элементов разработать простые диалоговые окна, которые, например, показаны в следующем разделе, и разработать одну программу средней сложности, управляю-

щую диалоговыми окнами и вызывающую требуемые функции ввода данных и рисования элементов. В XML-меню вызывать универсальную программу с требуемыми аргументами.

В-третьих, можно для всех элементов разработать собственные короткие программы, по возможности использующие универсальные функции ввода данных и рисования. В XML-меню можно вызывать необходимые программы с требуемыми аргументами.

Первый вариант следует отбросить сразу — одни недостатки при отсутствии достоинств. Второй вариант реализован нами в программе ru_san_pipe_detail. Несколько лет назад при разработке этой программы мы польстились на красоту решения с передачей в аргументах имени функции, но оказалось, что и с программой средней сложности трудно разбираться, если не заниматься исключительно ей.

Начиная разработку "с нуля", лучше действовать по третьему варианту — он оптимален и для программиста, и для пользователя. Вместо одной сложной программы мы будем иметь много, но простых. Наш опыт показывает, что разбираться с маленькими программами намного легче, особенно имея такой инструмент, как программа ruLispExplorer *(см. главу 9)*.

Организация ввода данных

В *главе 9* мы уже упоминали об организации хранения данных — как раз на примере рисования трубопроводов. Для каждого элемента нам, как минимум, потребуется выбор одного диаметра (рис. 29.14), диаметра и наименования изделия (рис. 29.15), иногда двух диаметров (рис. 29.16), а иногда — диаметра и иного параметра, например, радиуса изгиба (рис. 29.17).

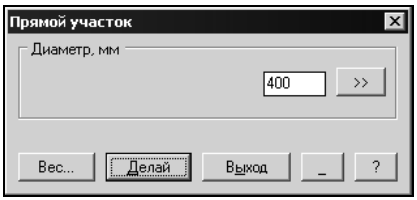


Рис. 29.14. Диалоговое окно ввода диаметра

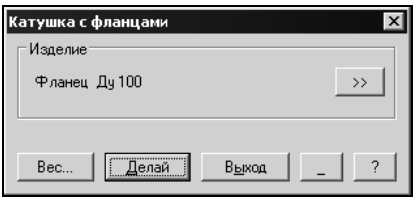


Рис. 29.15. Диалоговое окно ввода наименования изделия и диаметра

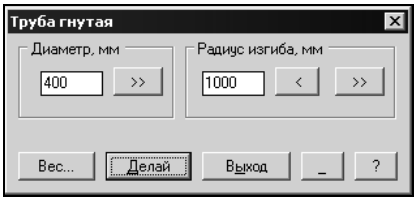


Рис. 29.16. Диалоговое окно ввода диаметров магистрали и ответвления

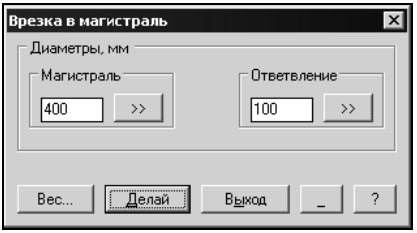


Рис. 29.17. Диалоговое окно ввода диаметра и радиуса изгиба

Для подписывания диаметров мы уже предусмотрели ввод текста диаметра в функции `ru-get-diam-text` (см. листинг 29.9). Выбор диаметров производился из меню options\\dia_lst.xml (листинг 29.12). В этом меню содержатся *строки* "всего круглого", и для рисования это меню не подходит — рядом могут быть строки `500` (преобразуется в число) и `530x7` (в число не преобразуется).

Листинг 29.12. Фрагмент файла dia_lst.xml

```
<?xml version='1.0' encoding='windows-1251'?>
<root name='Изделия с диаметрами' sdata=''>
  <item name='Диаметры условных проходов' sdata=''>
    <item name='Ду  15' sdata='15'/>
    <item name='Ду  20' sdata='20'/>
...
    <item name='Ду 1000' sdata='1000'/>
    <item name='Ду 1200' sdata='1200'/>
  </item>
  <item name='Трубы' sdata=''>
    <item name='Водогазопроводные' sdata=''>
      <item name='Обыкновенные' sdata=''>
        <item name='Труба 15x2.8 ГОСТ 3262-75' sdata='15x2.8'/>
...
        <item name='Труба 65x4   ГОСТ 3262-75' sdata='65x4'/>
      </item>
...
    </item>
    <item name='Электросварные' sdata=''>
      <item name='Обычные' sdata=''>
        <item name='Труба 32x2.5 ГОСТ 10704-76' sdata='32x2.5'/>
...
        <item name='Труба 530x7 ГОСТ 10704-76' sdata='530x7'/>
      </item>
    </item>
  </item>
  <item name='Воздуховоды' sdata=''>
    <item name='Воздуховод D100' sdata='100'/>
...
    <item name='Воздуховод D2000' sdata='2000'/>
  </item>
  <item name='Сталь' sdata=''>
    <item name='Пруток 5' sdata='5'/>
...
    <item name='Пруток 36' sdata='36'/>
  </item>
</root>
```

Для разных элементов трубопроводов требуются различные параметры, связанные с *диаметром условного прохода* трубы. Если для прямого участка необходим только диаметр условного прохода[1], то для рисования фланца, кроме диаметра условного

[1] Трубопроводы мы рисуем по диаметру условного прохода, пренебрегая толщиной стенки, иначе выбор будет слишком велик при отсутствии потребности в такой точности.

прохода нужны уже четыре параметра, да еще зависящие от расчетного давления для фланца. Для расчетных программ могут потребоваться и дополнительные параметры. От идей, рассматривавшихся в *главе 9*, мы давно перешли к реализации. Мы уже умеем работать с базами данных и вообще могли бы занести все параметры всех марок и типоразмеров труб в базу данных и для разных целей использовать требуемые данные. Могли бы, но не стали этого делать, т. к. это сильно осложнило бы и разработку программ, и работу пользователей, да и работающую систему мы просто не успели бы сделать. Пользователям нужны не идеи, а готовые программы. Данные трубопроводов, за исключением преходящих номеров стандартов, не меняются десятилетиями, и работа с базой данных в таких условиях не имеет смысла.

Хранение данных мы предусмотрели в XML и пока об этом не жалеем. Но и здесь имеются нюансы. Примеры выбора данных по диаметрам из XML-меню мы приводили в *главе 16* и упоминали о необходимости совместимости со старыми программами системы BestIA, в которой для хранения данных использовалась одна строка и в которой данные разделялись точкой с запятой. Это были файлы формата BTR, преобразованные теперь в XML. Строка данных содержалась в атрибуте с именем `SDATA` (листинг 29.13).

Листинг 29.13. Фрагмент файла diam_nominal.xml

```
<?xml version='1.0' encoding='windows-1251'?>
<root name='Диаметры условных проходов' sdata=''>
  <item name='Ду 15' sdata='15.0;95.0;65.0;14.0;47.0;2.0;33.0'/>
  <item name='Ду 20' sdata='20.0;105.0;75.0;14.0;58.0;2.0;34.0'/>
...
  <item name='Ду 500' sdata='500.0;730.0;660.0;44.0;615.0;4.0;100.0'/>
  <item name='Ду 600' sdata='600.0;840.0;770.0;49.0;720.0;5.0;115.0'/>
...
</root>
```

В строке `SDATA` содержатся размеры трубопровода и фланцев (для одного из давлений). Что именно означает каждое число, когда-то знали только мы, но уже забыли. Разобраться с этим можно только анализируя программы. В программе требуемый параметр извлекается по номеру элемента в списке, что также затрудняет понимание смысла. Но XML позволяет хранить данные в именованных атрибутах (листинг 29.14). Такой формат данных понятен любому специалисту, знающему, что такое фланцы.

Листинг 29.14. Фрагмент файла flange.xml

```
<?xml version='1.0' encoding='windows-1251'?>
<flanges_base name='Фланцы' Ду='' Дфл='' Дболт='' Толщ='' Двыст='' Выступ=''
Патрубок='' Дотв='' n=''>
  <flange name='Приварной встык Ру 1.6 МПа' Ду='' Дфл='' Дболт='' Толщ='' Двыст=''
Выступ='' Патрубок='' Дотв='' n=''>
    <flange name='Фланец Ду 15' Ду='15.0' Дфл='95.0' Дболт='65.0' Толщ='12.0'
Двыст='47.0' Выступ='2.0' Патрубок='33.0' Дотв='14' n='4'/>
...
    <flange name='Фланец Ду 1000' Ду='1000.0' Дфл='1255.0' Дболт='1170.0'
Толщ='49.0' Двыст='1110.0' Выступ='5.0' Патрубок='110.0' Дотв='45' n='28'/>
  </flange>
```

```
  <flange name='Приварной встык Ру 2.5 МПа' Ду='' Дфл='' Дболт='' Толщ='' Двыст=''
Выступ='' Патрубок='' Дотв='' n=''>
    <flange name='Фланец Ду 15' Ду='15.0' Дфл='95.0' Дболт='65.0' Толщ='14.0'
Двыст='47.0' Выступ='2.0' Патрубок='33.0' Дотв='14' n='4'/>
    <flange name='Фланец Ду 1000' Ду='1000.0' Дфл='1315.0' Дболт='1210.0'
Толщ='59.0' Двыст='1140.0' Выступ='5.0' Патрубок='150.0' Дотв='56' n='28'/>
  </flange>
</flanges_base>
```

Выбор из XML-меню с помощью вызова функции

```
(setq lst (car (ru-xml-get-att-list "Выбор диаметра"
  (ru-file-menu-xml "options\\flanges.xml")
  (list "name" "Ду" "Дфл" "Дболт" "Толщ" "Двыст" "Выступ" "Патрубок"
   "Дотв" "n") "Ду")))
```

может вернуть удобный для чтения ассоциированный список

```
(("name" . "Фланец Ду 15") ("Ду" . "15.0") ("Дфл" . "95.0")
  ("Дболт" . "65.0") ("Толщ" . "12.0") ("Двыст" . "47.0")
  ("Выступ" . "2.0") ("Патрубок" . "33.0") ("Дотв" . "14") ("n" . "4")).
```

LISP позволяет найти значение из ассоциированного списка и даже присвоить его переменной, имя которой является ключом для поиска:

```
(setq var_name "Дфл")
(set (read var_name)(atof (car (ru-list-massoc var_name lst))))
(print Дфл)
145.0
```

Все это очень хорошо, но требует и переделки наших программ, и переписывания старых XML-меню. Себе мы работы добавим, сроки сорвем, а заказчик наших улучшений и не заметит — это внутренние детали, не интересующие пользователей, и им нет дела до того, как мы анализируем списки.

Рассмотрим еще один вариант. Попробуем вообще избавиться от диалоговых окон, даже таких простых, как на рис. 29.14—29.17. Пока наш пользователь работает в многостадийной системе, включающей, как минимум:

- выбор элемента трубопровода (рис. 29.13);
- выбор типоразмера (рис. 29.14—29.17);
- рисование;
- возврат в диалог выбора типоразмера;
- выбор другого типоразмера и повторное рисование;
- выход и возврат в меню выбора элементов;
- выход в систему AutoCAD.

Если мы совместим меню элементов с типоразмерами, то пользователь будет иметь дело с единственным диалоговым окном, а мы будем писать только короткие функции рисования конкретного элемента с конкретными параметрами. Вместо переработки и программ, и меню мы будем иметь дело с написанием новых коротких программ и новых макросов меню. Написать нужно около десятка новых простых

функций, а это легче, чем переделать три больших и сложных программы. Пользователи также получат реальную пользу от ускорения работы. Вот этот вариант мы и примем для дальнейшей работы.

Универсальная функция рисования элементов труб

Первым делом разработаем универсальную функцию рисования любого элемента трубопроводов (листинг 29.15). Аргументами функции являются:

- имя функции рисования;
- список аргументов для функции рисования (в частном случае это может быть не список, а атом);
- сообщение, выводимое при запросе первой точки;
- сообщение, выводимое при запросе второй точки.

Листинг 29.15. Функция `ru-pipe-draw-any`

```
(defun ru-pipe-draw-any (func param_lst msg_1pt msg_2pt / start_pnt end_pnt end)
 (ru-app-begin)
 (while (not end)(setq start_pnt (if *ru_last_end_point*
  (ru-get-point-or-lw-or-continue-or-exit msg_1pt)
  (ru-get-point-or-lw-or-exit msg_1pt)))
  (cond
    ((ru-is-point start_pnt)
     ((eval func) start_pnt (ru-get-point-reguired msg_2pt start_pnt)
       param_lst))
    ((null start_pnt) (setq end t)))) (ru-app-end) (princ))
```

И эта функция будет рисовать все элементы трубопроводов? Конечно. Куда же она денется, будет рисовать, и не только элементы трубопроводов, но и воздуховодов, и вообще всего, что можно нарисовать, имея две точки и список параметров. Теперь мы можем избавиться от сложной программы ru_san_pipe_detail и от всех больших и малых диалоговых окон. Работа теперь сводится к разработке простых функций и наполнению их вызовами XML-меню. Приведем пример самой простой функции — рисования прямого участка трубы или концентричного перехода.

Листинг 29.16. Функция `ru-pipe-line`

```
(defun ru-pipe-line (pnt_start pnt_end param_lst / r_start r_end ang right left)
  (setq r_start (/ (car param_lst) 2.0) r_end   (/ (cadr param_lst) 2.0)
    ang (angle pnt_start pnt_end) right (ru-geom-go-right ang)
    left (ru-geom-go-left ang))
  (ru-pline-add (list (polar pnt_start right r_start)
    (polar pnt_end right r_end) (polar pnt_end left r_end)
          (polar pnt_start left r_start)) T 0 (ru-lw-current) nil)
  (ru-line-add pnt_start (setq *ru_last_end_point* pnt_end) 0 "DASHDOT2")
)
```

Для того чтобы не сбивать себя с толку вызовом такой функции со списком двух одинаковых диаметров для рисования простой трубы, сделаем еще упрощенный вариант (листинг 29.17).

Листинг 29.17. Функция ru-pipe-simple

```
(defun ru-pipe-simple (pnt_start pnt_end param_lst)
  (ru-pipe-line pnt_start pnt_end (list param_lst param_lst)))
```

Рисование концентричных переходов выполняется функцией, приведенной в листинге 29.18.

Листинг 29.18. Функция ru-pipe-reducer-concentr

```
(defun ru-pipe-reducer-concentr (start_pnt end_pnt param_lst)
  (ru-pipe-line start_pnt
    (polar start_pnt (angle start_pnt end_pnt) (nth 1 param_lst))
    (list (nth 0 param_lst) (nth 1 param_lst))))
```

Еще один пример — рисование гнутой трубы (листинг 29.19), для которой требуется задавать минимальный радиус гиба.

Листинг 29.19. Функция ru-pipe-line-bend

```
(defun ru-pipe-line-bend (first_point second_point param_lst / diam_main rad_mid
ang center_line list_point vla_array_objs)
 (setq diam_main (/ (ru-conv-millimeter-to-unit (nth 0 param_lst)) 2.0)
    rad_mid (nth 1 param_lst) ang (angle first_point second_point))
 (if (setq list_point (ru-trass-draw-lw-from-pt3-min-segm
       "гнутой трубы" first_point second_point 0 0 rad_mid))
  (progn
    (setq center_line (entlast))
    (command "_.FILLET" "_R" (ru-conv-millimeter-to-unit rad_mid))
    (command "_.FILLET" "_P" center_line)
    (if (setq vla_array_objs
        (ru-obj-ent-offset center_line (+ diam_main)))
      (ru-obj-vla-array-mod vla_array_objs "LineWeight" (ru-lw-current)))
    (if (setq vla_array_objs
        (ru-obj-ent-offset center_line (- diam_main)))
      (ru-obj-vla-array-mod vla_array_objs "LineWeight" (ru-lw-current)))
    (if (ru-ltype-load "DASHDOT2")(ru-ent-mod center_line "DASHDOT2" 6))
 ))(princ))
```

Для рисования трубопроводной арматуры используется специальная функция (листинг 29.20). Реалистичные изображения арматуры выполнены в виде библиотечных блоков (виды сбоку, сверху и вдоль оси трубы). Для того чтобы не рисовать для каждого конструктивного исполнения и диаметра отдельные блоки, применен прием формирования "конечных" блоков из промежуточных единичных деталей. В функцию рисования арматуры передается список параметров из имени библиотеки, имени блока и строительной длины.

Листинг 29.20. Функция ru-pipe-valve-details

```
(defun ru-pipe-valve-details (start_pnt end_pnt param_lst / ang block_name scale)
 (setq block_name (nth 1 param_lst)
    scale (ru-conv-millimeter-to-unit 1) ang (angle start_pnt end_pnt)
    *ru_last_end_point* (polar start_pnt ang
    (ru-conv-millimeter-to-unit (nth 2 param_lst))))
 (if (ru-block-lib-insert (nth 0 param_lst) block_name)
  (progn
    (ru-block-insert-obj block_name start_pnt scale scale scale ang)
    (if (not
     (ru-get-is-point-right-by-axis "верха изделия" start_pnt end_pnt))
        (ru-obj-ent-ss-mirror (entlast) start_pnt end_pnt t)))))
```

После всех изменений мы можем откорректировать XML-меню, характерные пункты которого показаны в листинге 29.21.

Листинг 29.21. Фрагменты файла pipe_detail.xml

```
<?xml version='1.0' encoding='windows-1251' ?>
<root name='Трубопроводы стальные в 3 линии' macro=''>
<item name='Арматура'  macro=''> <item name='Вид сбоку'  macro=''>
<item name='Задвижки'  macro=''><item name='Чугунные' macro=''>
...
 <item name='Ду 400*' image='TRUBA\ARMREAL\30CH_B1.GIF' comment='30ч6бр'
 macro='(ru-pipe-draw-any (quote ru-pipe-valve-details) (list "ru-lib-valve-detail"
"RU_30ch6br_400" 600) "Начало изделия" "Направление второго конца")'/>
...
 </item> </item></item>
<item name='Трубы' macro=''> <item name='Прямой участок' macro=''>
 <item name='Ду 15' image='TRUBA\PIPES\PIPE.GIF'
comment='Прямой участок трубы'
macro='(ru-pipe-draw-any (quote ru-pipe-simple) 15.0 "Начало участка трубы"
"Конечная точка участка трубы")'/>
...
</item>
<item name='Труба гнутая' macro=''>
 <item name='Ду  15' image='TRUBA\PIPES\PIPE_GIB.GIF'
comment='Гнутый участок трубопровода'
macro='(ru-pipe-draw-any (quote ru-pipe-line-bend)(list   15.0 50.0) "Начало гнутой
трубы" "Вторая точка гнутой трубы")'/>
...
</item>
 <item name='Переходы стандартные' macro=''>
<item name='Концентричные' macro=''>
...
 <item name='100x200' image='TRUBA\PIPES\PERX_KR.GIF'
comment='Переход концентричный'
macro='(ru-pipe-draw-any (quote ru-pipe-reducer-concentr)
(list 100.0 200.0 95.0)"Начало перехода" "Направление потока")'/>
...
 </item> </item> </item></root>
```

Программы для рисования вентиляции

Программы для рисования рабочих чертежей вентиляционных систем (планы, установки, схемы) принципиально не отличаются от программ для рисования трубопроводов. Проблемы и пути их решения точно такие же. Некоторая сложность есть только в функциях рисования отдельных элементов, например, тройников "аспирационного" типа, и немного усложняет работу использование круглых и прямоугольных воздуховодов и различных вариантов соединений (на фланцах, бандажах, рейках, термоусаживающихся манжетах). Разбираться с такими программами мы не будем, приведем только пример готового рисунка (рис. 29.18), в котором все вентиляционное оборудование нарисовано с помощью функций системы ruCAD.

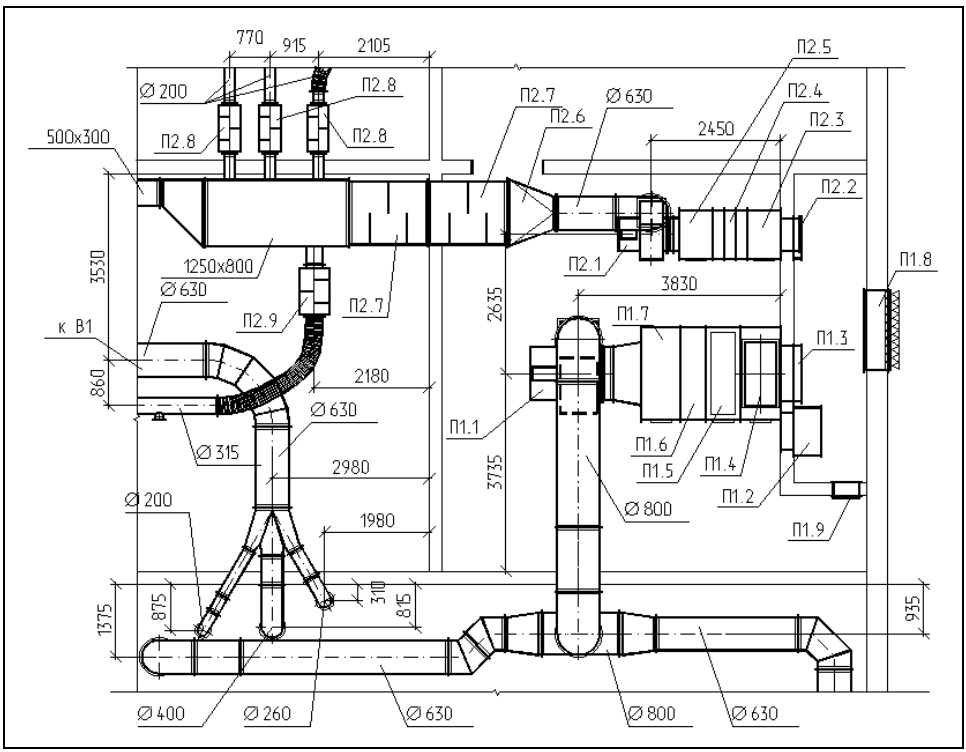


Рис. 29.18. Фрагмент плана вентиляции

Кстати, рис. 29.18 демонстрирует применяемое нами много лет решение — отказ от разработки чертежей установок систем с "угловыми" спецификациями и проставление на общем плане позиционных обозначений по спецификации оборудования. В большом проекте такой прием позволяет избавиться от десятков никому не нужных листов.

Резюме

Мы рассмотрели только основные программы для сантехнических разделов проекта — самые "ходовые" и самые принципиальные. Разумеется, требуется еще много других средств автоматизации рисования сантехники (и они есть), но большую часть изображений можно создавать с использованием функций, разобранных в книге. Заглядывайте в другие "специальные" главы и находите там решения, необходимые сантехникам. А если чего-то не найдете в книге или на компакт-диске, заходите на сайт книги[1] — там мы будем выкладывать дополнения к системе.

[1] **www.kurganobl.ru/cad**.

ГЛАВА 30

Вывод чертежа на бумагу

До сих пор основной продукцией проектных организаций являются чертежи, выведенные на бумажный носитель. По чертежам, выведенным на бумагу[1], оценивается вся работа проектировщиков. Процесс печати в системе AutoCAD от версии к версии все более усложняется, но фирму в этом упрекнуть нельзя — сложность работы определяется усложнением задач. О том, как печатать чертежи стандартными методами, мы писать не будем — это тема других изданий[2]. Остановимся только на возможности программирования этого процесса.

Программы, автоматизирующие печать, встречаются редко и решают они обычно вспомогательные задачи, например учет отпечатанных чертежей. В частных случаях, при постоянных условиях печати, можно автоматизировать и сам процесс.

Мы попробуем решить одну из актуальных задач, не решенных в самой системе AutoCAD — вывод больших чертежей на малоформатные устройства. "Сытый голодного не разумеет" — гласит пословица. Видимо поэтому в фирме Autodesk не предусмотрели такое необходимое усовершенствование. "Там" это решается просто — хочешь иметь большие чертежи — покупай соответствующее оборудование. Российские разработчики ближе к нашим реалиям, поэтому российская система КОМПАС-ГРАФИК выгодно отличается тем, что имеет возможность печати чертежей по частям.

Замечание

Не всегда печать по частям требуется "бедному студенту" для распечатки дипломного проекта на принтере офисного класса. Нам приходилось делать это, имея плоттер формата A0 — требовалось отпечатать общий план города, который наклеивался на планшет размером 6×3 метра.

Конечно, из системы AutoCAD можно отпечатать чертеж по частям с использованием различных "шаманских" приемов, не являющихся темой нашей книги, но мы

---

[1] Процесс вывода на бумажный носитель (или в файл любого формата, например, DWF) в дальнейшем будем именовать печать, независимо от того, на какое физическое устройство выводится чертеж.

[2] Например, *см. главу 43* книги Н. Полещука "AutoCAD 2004", изд. "БХВ-Петербург", 2004. — 976 с. (серия "В подлиннике").

попробуем выполнить это программным путем. Пример решения такой задачи показала Надежда Толстоба[1], разработавшая "для студентов" программу печати на принтер формата A4. Ее программа решает задачу, но в частных условиях. Мы попробуем выполнить ее на профессиональном[2] уровне, попутно разобравшись с массой интересных вопросов.

Алгоритм печати

В целом *алгоритм печати по частям* кажется простым — выделять рамкой фрагмент рисунка, соответствующий формату бумаги, отправлять на устройство печати и переходить к следующему фрагменту. Реализовать это практически гораздо сложнее, чем придумать. Решать приходится много частных подзадач. Рассмотрим процесс печати с точки зрения пользователя нашей будущей программы. Для упрощения программирования мы не будем разрабатывать сложное диалоговое окно, а воспользуемся простыми средствами, предусмотрев линейную структуру программы. На каждом шаге зафиксируем задачи, которые предстоит решить программисту.

Шаг 1. Выбор компоновки

Пользователь выбирает компоновку (Layout), выводимую на печать (рис. 30.1).

Для программиста здесь сложностей нет. Переключение между компоновками у нас уже решено. В принципе, этот шаг не нужен (печатать надо всегда активную компоновку), но слишком часто пользователи, глядя на модель, думают, что печататься будет какой-нибудь Layout3.

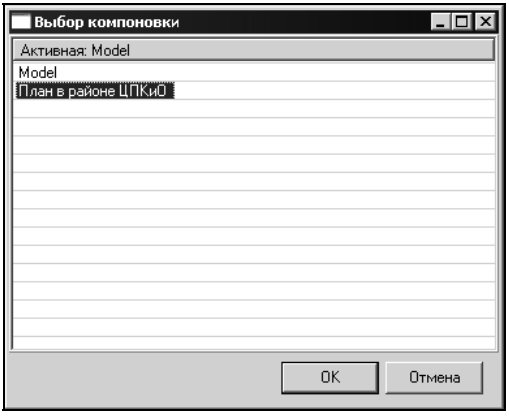


Рис. 30.1. Выбор компоновки для печати

[1] **aco.ifmo.ru/~nadinet/**.

[2] Не подвергая ни малейшему сомнению профессионализм Надежды Дмитриевны как преподавателя. Кстати, именно на ее сайте начинающие LISP-программисты могут найти не столь "заумные", как в нашей книге, материалы для начального обучения.

Шаг 2. Выбор устройства печати

Пользователь выбирает устройство печати (рис. 30.2).

Проблемы программиста заключаются в получении списка доступных устройств печати, а их может быть много, в том числе сетевые и виртуальные.

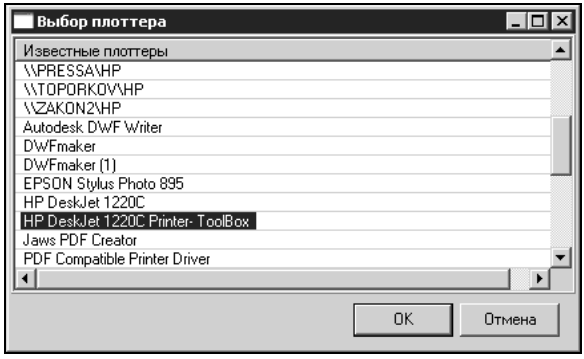


Рис. 30.2. Выбор устройства печати

Шаг 3. Выбор формата бумаги

Пользователь выбирает формат (размер) бумаги, поддерживаемой выбранным устройством (рис. 30.3). При выводе в растровый формат вместо размеров бумаги будет выбираться размер изображения в пикселах.

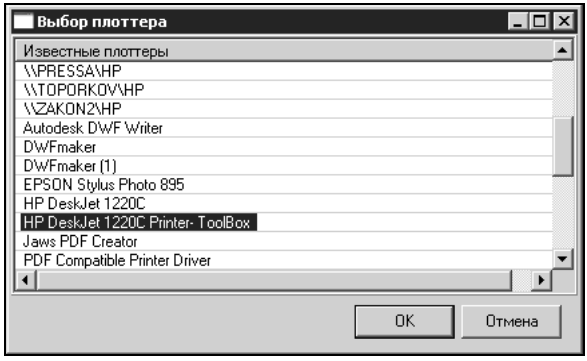


Рис. 30.3. Выбор формата бумаги

Для программиста возникают сложности — у каждой компоновки может быть свой набор поддерживаемых стандартов бумаги. Их надо выяснить и вывести для выбора. Но не просто получить список поддерживаемых форматов, а учесть, что у каждого формата имеется имя, показываемое пользователю, каноническое имя, используемое программами, длина и ширина листа, а также код формата.

Шаг 4. Выбор стиля печати

Программа предлагает выбрать стиль печати (рис. 30.4).

Программист должен позаботиться об использовании выбранного стиля и о варианте отказа от выбора стиля.

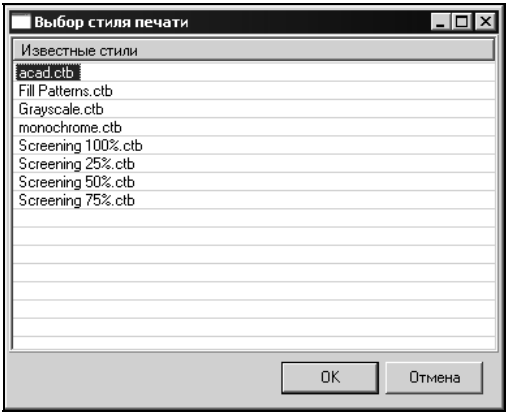


Рис. 30.4. Выбор стиля печати

Шаг 5. Выбор зоны печати

Программа делает активной выбранную компоновку и запрашивает:

Укажи первый угол зоны печати [Extents/Display/Limits]<Cancel>:
Укажи второй угол зоны печати:

Пользователь указывает зону печати.

Для этого шага программист должен предусмотреть, с учетом всех выбранных ранее параметров, разбивку зоны печати на "конверты" с учетом формата бумаги, сформировать массив зон печати, нарисовать эти зоны на экране и дать возможность передвигать созданный массив.

Шаг 6. Формирование массива листов

Программа формирует массив листов (рис. 30.5) и предлагает пользователю уточнить его расположение. Границы листов показываются специальными контурами ("конвертами").

В ответ на запрос:

Точное расположение массива листов <Как есть>:

Пользователь может передвинуть мышью весь массив листов или согласиться с предложенной "нарезкой".

Программист должен предусмотреть возможность передвижения массива листов.

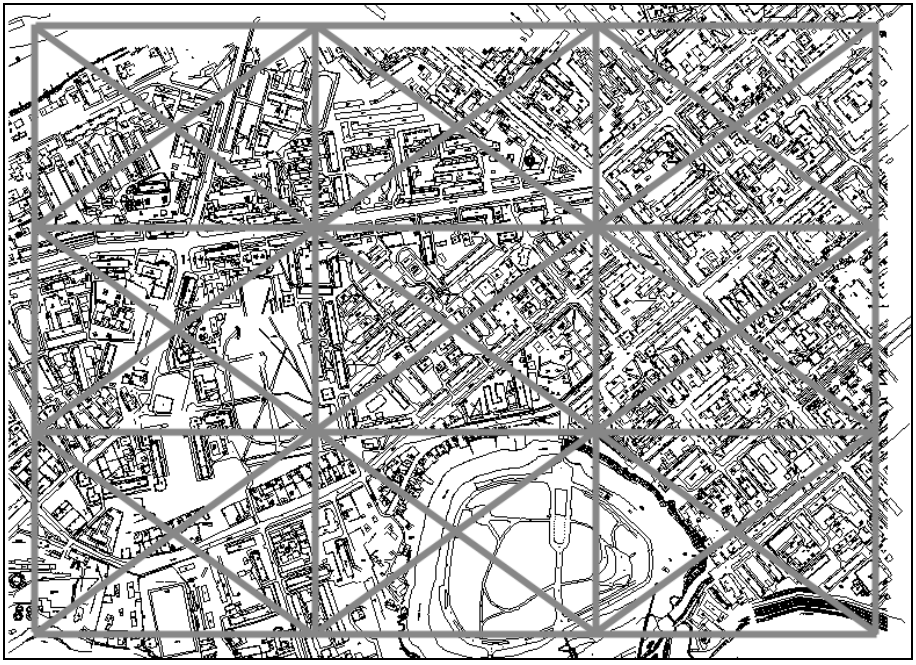


Рис. 30.5. Границы массива листов

Шаг 7. Печать

Программа сообщает:

Подготовлено листов: 9. Печатать? [Yes/No/Selected]<No>:

При выборе опции **Selected** будет подсвечиваться "конверт" очередного листа и выводиться запрос:

Печатать подсвеченный лист? [Yes/No/Preview]<Yes>:
Effective plotting area: 201.61 wide by 279.84 high

При выборе опции **Preview** выводится стандартное окно предварительного просмотра печати. При согласии на печать происходит вывод листа на устройство, а "конверт" отпечатанного листа исчезает.

Программист должен все это реализовать в кодах, да еще при условии, что реально проверить работу невозможно — кто же ему позволит экспериментировать с реальной бумагой и устройствами?

Реализация алгоритма печати

Приступаем к реализации задуманного. Для этого нам необходимо хорошо изучить объектную модель печати в системе AutoCAD.

Как выполняется печать

Печатать следует с использованием объекта `Plot`, имеющего, помимо прочих, интересующие нас методы:

- `PlotToDevice` — с опциональным параметром `PlotConfig`;
- `PlotToFile` — с обязательным параметром `PlotFile` и опциональным параметром `PlotConfig`.

Параметр `PlotConfig` должен содержать полное имя PC3-файла с требуемой конфигурацией. Если этот параметр не задан, используется текущая конфигурация. Отсюда вытекает, что для полного управления процессом печати нам необходимо подготовить правильную конфигурацию печати. Обычно параметры конфигурации создаются с помощью инструмента **Autodesk Plotter Manager**, конфигурация для компоновки — с помощью диалогового окна команды PAGESETUP (ПАРАМЛИСТ). Нам предстоит проделать это программным путем.

Замечание

А почему бы не выполнять все настройки с помощью команды PAGESETUP (ПАРАМЛИСТ), в которой пользователь настроит все, как ему надо? Да, можно было бы, если бы в диалоговом окне, вызываемом по этой команде, не было кнопки **Plot**. Нам нужно решить задачу вывода по частям, а не печати "как влезет". Конечно, можно бы и предупредить пользователя, чтобы он не нажимал на **Plot**, но ведь нажмет обязательно. Возможна и нудная разработка собственного диалогового окна, но мы пока ограничимся установкой набора типовых параметров, которые считаем оптимальными.

Не будем использовать и команду –PLOT, т. к. предугадать все возможные опции для различных устройств практически невозможно.

Объект конфигурации `PlotConfiguration` имеет множество свойств и методов. Мы будем работать с основными:

- `ConfigName` — имя PC3-файла или устройства печати;
- `CanonicalMediaName` — каноническое имя бумаги;
- `CenterPlot` — центрировать ли печать (мы будем центрировать всегда);
- `PaperUnits` — код единиц измерения размеров бумаги (мы принимаем `acMillimeters`);
- `PlotHidden` — печатать ли скрытые объекты (мы будем печатать);
- `PlotOrigin` — смещение начала зоны печати;
- `PlotRotation` — поворот бумаги при печати;
- `PlotType` — тип печати (мы будем всегда использовать `acWindow` — таково назначение нашей программы);
- `PlotViewPortBorders` — печатать ли границы виртуальных экранов (мы не будем);
- `PlotViewPortsFirst` — печатать ли сначала содержимое виртуальных экранов;
- `PlotWithLineweights` — печатать ли с учетом веса линий в стиле печати (мы будем печатать с учетом веса линий в рисунке);
- `PlotWithPlotStyles` — использовать ли стили печати, назначенные объектам;
- `StyleSheet` — имя таблицы стилей.

Вывести чертеж на печать очень просто — достаточно выполнить LISP-выражение `(vla-plottofile plot_obj file_name device)` или `(vla-plottodevice plot_obj device)`. Вся остальная невидимая часть "айсберга" программы должна подготовить конфигурации объекта печати и обеспечить пользовательский интерфейс. Как это делается, мы разберем в следующих разделах.

Основная функция печати

Сначала разберем главную функцию печати (листинг 30.1), а затем вспомогательные функции.

Листинг 30.1. Функция `ru-plot-array`

```
(defun ru-plot-array (/plot_obj array conf cs1 cs2 cur_plot cur_ucs dpi layer
layout_list layout_obj master p_size paper_canonic_name paper_cfg paper_code
paper_height paper_name paper_width plotter_papers plot_device plot_file plot_scale
p_rot style tmp_ucs)
;|-------------------------------------------------------------------------
 1. Выбор компоновки
-------------------------------------------------------------------------|;
  (if (setq layout_list (ru-dlg-select-layout))
   (progn
     (setq layout_obj (car (cadr layout_list)))
;|-------------------------------------------------------------------------
 Установка выбранной компоновки
-------------------------------------------------------------------------|;
      (vla-put-activelayout (ru-obj-get-active-document) layout_obj)
;|-------------------------------------------------------------------------
 2. Выбор устройства
-------------------------------------------------------------------------|;
      (if (setq plot_device (ru-dlg-select-plot-device layout_obj))
        (progn
;|-------------------------------------------------------------------------
Список конфигураций бумаги для выбранного устройства
-------------------------------------------------------------------------|;
          (setq plotter_papers
                (ru-plot-get-plotter-papers plot_device layout_obj T)
         );_ end of setq
;|-------------------------------------------------------------------------
Выбор конфигурации бумаги для устройства
-------------------------------------------------------------------------|;
          (if
            (setq paper_cfg (ru-dlg-select-papers-cfg plotter_papers))
             (progn
               (setq paper_canonic_name (nth 0 paper_cfg)
                     paper_name         (nth 1 paper_cfg)
                     paper_code         (nth 2 paper_cfg)
                     paper_width        (car (nth 3 paper_cfg))
                     paper_height       (cadr (nth 3 paper_cfg))
```

```
;|-----------------------------------------------------------------------
Расчет масштаба печати. Для пространства модели выполняется с учетом масштаба
и наименования единиц рисунка, устанавливаемого в свойствах рисунка.
Для пространства листа масштаб 1:1
-----------------------------------------------------------------------|;
                    plot_scale
                    (if (= (strcase (car layout_list)) "MODEL")
                       (ru-conv-millimeter-in-paper-to-unit 1.0)
                                  1
                    );_ end of if
               );_ end of setq
;|-----------------------------------------------------------------------
Запрос на печать в файл
-----------------------------------------------------------------------|;
     (if (ru-yes "Печатать в файл")
      (setq plot_file (ru-file-dwgname-type ""))
    );_ end of if
;|-----------------------------------------------------------------------
Показываем размеры бумаги и запрашиваем поворот. Это место самое "скользкое", т. к.
устройства по-разному реагируют на команды развернуть чертеж. Некоторые чертеж
разворачивают, а бумагу - нет
-----------------------------------------------------------------------|;
  (if (ru-yes (strcat "Устройство " plot_device
     "\nРазмер бумаги: Длина " (rtos paper_width) ", Высота "
       (rtos paper_height) "\nПовернуть на 90 градусов")
                 )
                (setq p_rot ac90degrees)
                (setq p_rot ac0degrees)
 );_ end of if
;|-----------------------------------------------------------------------
Выбор стиля печати
-----------------------------------------------------------------------|;
  (setq style
       (cond
          ((ru-dlg-select-style layout_obj))
          (T "")
       );_ end of cond
 );_ end of setq

;;;
;|-----------------------------------------------------------------------
Начало обработки. Ставим метку отката
-----------------------------------------------------------------------|;
    (vla-startundomark (ru-obj-get-active-document))
;|-----------------------------------------------------------------------
Устанавливаем временную систему координат
-----------------------------------------------------------------------|;
    (setq *ru-object-error-messages* nil
          cur_ucs (vla-get-activeucs (ru-obj-get-active-document))
          tmp_ucs_name "ru-temp-wcs"
   );_ end of setq
```

```
   (if (setq tmp_usc (ru-obj-collection-item-by-name
             "UserCoordinateSystems" tmp_ucs_name))
;|-------------------------------------------------------------------
Если временная система координат уже есть, а это могло быть при сбое, то удаляем ее
-------------------------------------------------------------------|;
     (ru-obj-error-apply 'vla-delete (list tmp_ucs))
  );_ end of if
  (setq tmp_ucs
     (apply 'vla-add
      (append (list (vla-get-usercoordinatesystems
          (ru-obj-get-active-document)))
          (mapcar 'vlax-3d-point
                    '((0.0 0.0 0.0) (1.0 0.0 0.0) (0.0 1.0 0.0))
         );_ end of mapcar
         (list tmp_ucs_name)
      );_ end of append
     )
 );_ end of setq
  (vla-put-activeucs (ru-obj-get-active-document) tmp_ucs)
;|-------------------------------------------------------------------
Определение разрешения для вывода в растр. Пока тут заглушка, в дальнейшем можно
предусмотреть диалоговый вариант
-------------------------------------------------------------------|;
 (setq dpi (if (and (= 2 paper_code) (setq dpi 600))
         (if (= 1 (getvar "MEASUREMENT")) (cvunit 600 "mm" "inch")
        ;; _ на самом деле DPI -> DPMM
                              dpi))
;|-------------------------------------------------------------------
Преобразование масштаба
-------------------------------------------------------------------|;
       plot_scale (/ plot_scale
                     (cond (dpi)
                           ((= 0 (vla-get-paperunits layout_obj)) 25.4)
                            (1.0)
                     );_ end of cond
                  );_ end of /
;|-------------------------------------------------------------------
Расчет размера листа с учетом масштаба
-------------------------------------------------------------------|;
      p_size  (mapcar (function (lambda (x) (* x plot_scale)))
               (if (= p_rot ac0degrees)
                 (list paper_width paper_height)
                 (list paper_height paper_width)
              );_ end of if
             );_ end of mapcar
  );_ end of setq
;|-------------------------------------------------------------------
Формирование массива листов
-------------------------------------------------------------------|;
   (if (setq master
```

```
;|-------------------------------------------------------------------
Запрос зоны печати
-------------------------------------------------------------------|;
     (ru-plot-get-plotarea p_size
;|-------------------------------------------------------------------
Создание временного непечатаемого слоя
-------------------------------------------------------------------|;
                 (vla-get-name (setq layer (ru-layer-create-tmp)))
     );_ end of ru-plot-get-plotarea
    );_ end of setq
    (progn
;|-------------------------------------------------------------------
Визуальное уточнение расположения листов
-------------------------------------------------------------------|;
      (princ "\nТочное расположение массива листов <Как есть>: ")
      (vl-cmdf "_.CHANGE" master "" "" pause "")
;|-------------------------------------------------------------------
Установка конфигурации плоттера
-------------------------------------------------------------------|;
      (setq cur_plot (ru-plot-layout-cfg layout_obj
          (list plot_device
                paper_canonic_name
                1; centerplot true-false
                acmillimeters; paperunits
                nil; plothidden true-false
                nil; plotorigin - должен быть массив (точка)
                p_rot; plotrotation
                 acwindow; plottype
                 0; plotviewportborders true-false
                 :vlax-true; plotviewportsfirst  true-false
                 nil; plotwithlineweights true-false
                (if style :vlax-true); plotwithplotstyles
                 style; stylesheet
                  nil; viewtoplot - строка - имя вида для печати
         );_ end of list
        );_ end of ru-plot-layout-cfg
    );_ end of setq
;|-------------------------------------------------------------------
Запоминание масштаба в переменных
-------------------------------------------------------------------|;
   (vla-getcustomscale layout_obj 'cs1 'cs2)
;|-------------------------------------------------------------------
Вставка массива "конвертов"
-------------------------------------------------------------------|;
    (setq array (ru-block-minsert-to-array master))
;|-------------------------------------------------------------------
Получение объекта печати
-------------------------------------------------------------------|;
   (setq plot_obj (vla-get-plot (ru-obj-get-active-document))
```

```
;|-------------------------------------------------------------------------
Запрос на печать
-------------------------------------------------------------------------|;
        conf (ru-get-kword (strcat "Подготовлено листов: "
                (itoa (length array)) ". Печатать? ")
              "Yes No Selected" "No")
   );_ end of setq
   (if (or (= conf "Yes") (= conf "Selected"))
     (progn
;|-------------------------------------------------------------------------
Установка масштаба печати
-------------------------------------------------------------------------|;
        (vla-setcustomscale layout_obj 1.0 plot_scale)
;|-------------------------------------------------------------------------
Печать массива листов
-------------------------------------------------------------------------|;
     (mapcar (function (lambda (x)
;|-------------------------------------------------------------------------
Печать одного листа
-------------------------------------------------------------------------|;
      (ru-plot-unit-now layout_obj plot_device p_size x plot_obj
        (if plot_file (strcat plot_file "#"
                      (itoa (1+ (vl-position x array)))))
        (if (= conf "Selected") T)
      );_ end of ru-plot-unit-now
      )) array
     );_ end of mapcar
;|-------------------------------------------------------------------------
Восстановление масштаба
-------------------------------------------------------------------------|;
      (vla-setcustomscale layout_obj cs1 cs2)
    );_ end of progn
  );_ end of if
;|-------------------------------------------------------------------------
Удаление массива листов
-------------------------------------------------------------------------|;
   (mapcar 'vla-delete array)
;|-------------------------------------------------------------------------
Восстановление конфигурации
-------------------------------------------------------------------------|;
   (ru-plot-layout-cfg layout_obj cur_plot)
  );_ end of progn
  );_ end of if
;|-------------------------------------------------------------------------
Восстановление системы координат
-------------------------------------------------------------------------|;
  (vla-put-activeucs (ru-obj-get-active-document) cur_ucs)
;|-------------------------------------------------------------------------
Удаление временных системы координат и слоя
-------------------------------------------------------------------------|;
  (ru-obj-error-apply 'vla-delete (list tmp_ucs))
  (ru-obj-error-apply 'vla-delete (list layer))
    (vla-endundomark (ru-obj-get-active-document))
```

```
;;; Закрываем все скобки
))))))
(princ)
)
```

Далее приведем незнакомые вспомогательные функции в порядке их использования в основной.

Функция `ru-dlg-select-plot-device` (листинг 30.2) вернет имя выбранного устройства печати. Выбор производится из списка устройств, используемых в заданном VLA-объекте компоновки.

Листинг 30.2. Функция `ru-dlg-select-plot-device`

```
(defun ru-dlg-select-plot-device (layout_obj)
  (ru-dlg-single-list "Выбор плоттера" "Известные плоттеры"
    (ru-plot-get-plotters-name-for-layout layout_obj)
 );_ end of ru-dlg-single-list
)
```

Список плоттеров формируется функцией `ru-plot-get-plotters-name-for-layout` (листинг 30.3).

Листинг 30.3. Функция `ru-plot-get-plotters-name-for-layout`

```
(defun ru-plot-get-plotters-name-for-layout (layout_obj / lst
                                             _ru-is-.pc3 non)
 (defun _ru-is-.pc3 (str / len)
    (if (> (setq len (strlen str)) 3)
      (= ".PC3" (strcase (substr str (- len 3))))
   );_ end of if
);_ end of defun
  (setq lst (vlax-safearray->list (vlax-variant-value
              (vla-getplotdevicenames layout_obj))))
  (cond
    ((vl-position "None" lst)
     (setq lst (vl-remove "None" lst) non "None")
   )
    ((vl-position "Нет" lst)
     (setq lst (vl-remove "Нет" lst) non "Нет")
   )
    (T nil)
 );_ end of cond
  (cons non
        (append (acad_strlsort (vl-remove-if '_ru-is-.pc3 lst))
                (acad_strlsort (vl-remove-if-not '_ru-is-.pc3 lst))
       );_ end of append
 );_ end of cons
)
```

Список конфигураций бумаги для заданных устройства и компоновки формируется функцией `ru-plot-get-plotter-papers` (листинг 30.4).

Листинг 30.4. Функция `ru-plot-get-plotter-papers`

```
(defun ru-plot-get-plotter-papers (plotter layout force / current res tmp)
 (if (or (not (setq tmp (assoc plotter *ru-plot-plotters-cfg*)))
         force
     );_ end of or
   (progn
     (setq *ru-object-error-messages* nil
       current (ru-plot-layout-cfg layout
                 '(nil nil nil nil nil nil nil nil nil nil nil nil nil nil)
               );_ end of ru-plot-layout-cfg
     res (cond
          ((ru-obj-error-apply 'vla-put-configname (list layout plotter)))
          ((ru-plot-get-layout-papers-cfg layout))
          (T nil)
         );_ end of cond
    );_ end of setq
     (ru-plot-layout-cfg layout current)
     (if (and res (listp res))
         (setq *ru-plot-plotters-cfg*
                (if (not tmp)
                  (cons (cons plotter res) *ru-plot-plotters-cfg*)
                  (subst (cons plotter res) tmp *ru-plot-plotters-cfg*)
                );_ end of if
         );_ end of setq
      );_ end of if
       res
   );_ end of progn
     (cdr tmp)
 );_ end of if
)
```

Список свойств всех типов бумаги плоттера компоновки возвращает функция `ru-plot-get-layout-papers-cfg` (листинг 30.5).

Листинг 30.5. Функция `ru-plot-get-layout-papers-cfg`

```
(defun ru-plot-get-layout-papers-cfg (layout / current res tmp0 tmp1)
  (setq current (vla-get-canonicalmedianame layout)
      res
      (mapcar 'ru-plot-get-paper-cfg
        (setq tmp0
          (acad_strlsort (ru-plot-get-layout-papers-name layout)))
          (repeat (length tmp0) (setq tmp1 (cons layout tmp1)))
     );_ end of mapcar
 );_ end of setq
```

```
  (vl-remove-if
    (function (lambda (x / y1 y2) (setq x (cadddr x))
        (or (minusp (setq y1 (car x))) (minusp (setq y2 (cadr x)))
            (zerop y1) (zerop y2)
        );_ end of or
     );_ end of lambda
   );_ end of function
    (vl-remove-if 'null res)
 );_ end of vl-remove-if
)
```

Конфигурацию бумаги с заданными каноническим именем и компоновкой возвращает функция `ru-plot-get-paper-cfg` (листинг 30.6).

Листинг 30.6. Функция `ru-plot-get-paper-cfg`

```
(defun ru-plot-get-paper-cfg (paper_canonical_name layout / px py mx my margins
valid area ptype)
  (cond
    ((ru-obj-error-apply
       'vla-put-canonicalmedianame (list layout paper_canonical_name))
   )
    ((ru-obj-error-apply 'vla-put-plotrotation (list layout 0)))
    ((ru-obj-error-apply 'vla-getpapersize (list layout 'px 'py)))
    ((ru-obj-error-apply 'vla-getpapermargins (list layout 'mx 'my)))
;|Ну, тупые! В mx попадает left и bottom, а в my - right и top (по листу!)|;
    ((setq valid T))
 );_ end of cond
  (if valid
    (progn
      (setq margins
          (apply 'mapcar (cons 'list
                       (mapcar 'vlax-safearray->list (list mx my))))
            ptype   (vla-get-paperunits layout)
            area    (if (= 2 ptype)
                      (list px py)
                      (list (apply '- (cons px (car margins)))
                            (apply '- (cons py (cadr margins)))
                      );_ end of list
                    );_ end of if
     );_ end of setq
      (list paper_canonical_name
            (vla-getlocalemedianame layout paper_canonical_name)
            ptype
            (if (= 2 ptype)
              (mapcar 'fix area)
              area
            );_ end of if
     );_ end of list
   );_ end of progn
 );_ end of if
)
```

Список строк названий бумаги для заданной компоновки получаем с помощью функции `ru-plot-get-layout-papers-name` (листинг 30.7).

Листинг 30.7. Функция `ru-plot-get-layout-papers-name`

```
(defun ru-plot-get-layout-papers-name (layout)
  (vla-refreshplotdeviceinfo layout)
  (vlax-safearray->list
    (vlax-variant-value (vla-getcanonicalmedianames layout))
 );_ end of vlax-safearray->list
)
```

Диалоговый выбор конфигурации бумаги из списка выполняем с помощью функции `ru-dlg-select-papers-cfg` (листинг 30.8).

Листинг 30.8. Функция `ru-dlg-select-papers-cfg`

```
(defun ru-dlg-select-papers-cfg (plotter_papers / name paper_cfg)
  (defun _find-in-list (name layout_papers_cfg_list / result)
    (foreach x layout_papers_cfg_list
      (if (= (cadr x) name)
        (setq result (cons x result))
     );_ end of if
   );_ end of foreach
    result
 );_ end of defun
  (if (setq name
     (ru-dlg-single-list "Выбор формата бумаги" "Известные форматы"
                 (mapcar 'cadr plotter_papers)))
    (setq paper_cfg (car (_find-in-list name plotter_papers)))
 );_ end of if
  paper_cfg
)
```

Выбор стиля печати для заданной компоновки производится с помощью функции `ru-dlg-select-style` (листинг 30.9), а формирование списка стилей — функцией `ru-plot-get-styles-for-layout` (листинг 30.10).

Листинг 30.9. Функция `ru-dlg-select-style`

```
(defun ru-dlg-select-style (layout_obj)
  (ru-dlg-single-list "Выбор стиля печати" "Известные стили"
    (ru-plot-get-styles-for-layout layout_obj)
 );_ end of ru-dlg-single-list
)
```

Листинг 30.10. Функция `ru-plot-get-styles-for-layout`

```
(defun ru-plot-get-styles-for-layout (layout_obj)
  (acad_strlsort
```

```
    (vlax-safearray->list
      (vlax-variant-value (vla-getplotstyletablenames layout_obj))
    )
  )
)
```

Выбор зоны печати и формирование массива "конвертов", отображающих расположение будущих листов, производятся с помощью функции `ru-plot-get-plotarea` (листинг 30.11).

Листинг 30.11. Функция `ru-plot-get-plotarea`

```
(defun ru-plot-get-plotarea (paper-size layer / inp1 inp2 valid dx dy modx mody p1
p2 sx sy)
 (cond
  ((progn
   (initget 128 "Extents Display Limits")
   (vl-catch-all-error-p
     (setq inp1 (vl-catch-all-apply
   'getpoint
  '("\nУкажи первый угол зоны печати [Extents/Display/Limits]<Cancel>: ")
  );_ end of vl-catch-all-apply
 );_ end of setq
  )
);_ end of progn
 )
  ((not inp1))
  ((= inp1 "Extents")
   (setq inp1  (getvar "EXTMIN")
   inp2  (getvar "EXTMAX") valid T)
)
  ((= inp1 "Display")
     (setq inp1  (getvar "VSMIN")
           inp2  (getvar "VSMAX") valid T)
 )
  ((= inp1 "Limits")
     (setq inp1  (getvar "LIMMIN")
           inp2  (getvar "LIMMAX") valid T)
 )
  ((progn
    (initget 1)
       (not (vl-catch-all-error-p
              (setq
                inp2 (vl-catch-all-apply
                       'getcorner
                       (list inp1 "\nУкажи второй угол зоны печати: ")
                     );_ end of vl-catch-all-apply
              );_ end of setq
            );_ end of vl-catch-all-error-p
      );_ end of not
    );_ end of progn
```

```
     (setq valid T)
 )
  (T nil)
);_ end of cond
 (if valid
    (progn
      (setq p1   (list (apply 'min (list (car inp1) (car inp2)))
                       (apply 'min (list (cadr inp1) (cadr inp2)))
                 );_ end of list
            p2   (list (apply 'max (list (car inp1) (car inp2)))
                       (apply 'max (list (cadr inp1) (cadr inp2)))
                 );_ end of list
            dx   (- (car p2) (car p1))
            dy   (- (cadr p2) (cadr p1))
            sx   (float (car paper-size))
            sy   (float (cadr paper-size))
            modx (fix (/ dx sx))
            mody (fix (/ dy sy))
     );_ end of setq
      (if (> (rem dx sx) 0)
        (setq modx (1+ modx))
     );_ end of if
      (if (> (rem dy sy) 0)
        (setq mody (1+ mody))
     );_ end of if
;|-------------------------------------------------------------------
Вставка конвертов в виде массива анонимных блоков
-------------------------------------------------------------------|;
  (ru-block-minsert-unit-unnamed p1 paper-size layer (list modx mody))
  (entlast)
 );_ end of progn
);_ end of if
)
```

Листинг 30.12. Функция `ru-block-minsert-unit-unnamed`

```
(defun ru-block-minsert-unit-unnamed (point scale layer matrix)
;;; Вставка массива анонимных блоков-конвертов
    (entmake (append '((0 . "INSERT")
                       (100 . "AcDbBlockReference")
                       (62 . 256)
                       (50 . 0)
                     )
                     (list (cons 2 (ru-block-make-unit-unnamed))
                           (cons 10 point)
                           (cons 8
                                 (cond (layer)
                                       ((getvar "CLAYER"))
                                 );_ end of cond
                           );_ end of cons
```

```
                        (cons 41 (car scale))
                        (cons 42 (cadr scale))
                        (cons 70 (car matrix))
                        (cons 71 (cadr matrix))
                        (cons 44 (car scale))
                        (cons 45 (cadr scale))
                    );_ end of list
            );_ end of append
    );_ end of entmake
);_ end of defun
```

Отдельный анонимный блок-конверт создает функция `ru-block-make-unit-unnamed` (листинг 30.13), возвращающая "имя" этого блока и помещающая это имя в глобальную переменную `*ru-block-unit-unnamed*`.

Листинг 30.13. Функция `ru-block-make-unit-unnamed`

```
(defun ru-block-make-unit-unnamed (/ item)
    (if (or (not *ru-block-unit-unnamed*)
            (not (tblobjname "block" *ru-block-unit-unnamed*))
        );_ end of or
        (setq *ru-block-unit-unnamed*
                (foreach item '(((0 . "BLOCK")
                                 (100 . "AcDbBlockBegin")
                                 (2 . "*U0")
                                 (70 . 1)
                                 (10 0.0 0.0 0.0)
                                 (62 . 256)
                                )
                                ((0 . "LWPOLYLINE")
                                 (100 . "AcDbPolyline")
                                 (90 . 4)
                                 (70 . 1)
                                 (43 . 0.01)
                                 (38 . 0.0)
                                 (39 . 0.0)
                                 (10 0.0 0.0)
                                 (10 1.0 0.0)
                                 (10 1.0 1.0)
                                 (10 0.0 1.0)
                                 (210 0.0 0.0 1.0)
                                 (62 . 0)
                                )
                                ((0 . "LWPOLYLINE")
                                 (100 . "AcDbPolyline")
                                 (90 . 2)
                                 (70 . 0)
                                 (43 . 0.01)
                                 (38 . 0.0)
                                 (39 . 0.0)
```

```
                                (10 0.0 0.0)
                                (10 1.0 1.0)
                                (210 0.0 0.0 1.0)
                                (62 . 0)
                               )
                               ((0 . "LWPOLYLINE")
                                (100 . "AcDbPolyline")
                                (90 . 2)
                                (70 . 0)
                                (43 . 0.01)
                                (38 . 0.0)
                                (39 . 0.0)
                                (10 0.0 1.0)
                                (10 1.0 0.0)
                                (210 0.0 0.0 1.0)
                                (62 . 0)
                               )
                               ((0 . "ENDBLK")
                                (100 . "AcDbBlockEnd")
                               )
                              )
                        (entmake (append (list (car item)
                                               '(100 . "AcDbEntity")
                                         );_ end of list
                                         (append (cdr item)
                                                 '((8 . "0")
                                                   (67 . 0)
                                                   (410 . "Model")
                                                   )
                                         );_ end of append
                                 );_ end of cons
                        );_ end of entmake
                );_ end of foreach
        );_ end of setq
    );_ end of if
     *ru-block-unit-unnamed*
);_ end of defun
```

Для отрисовки "конвертов" функцией `ru-layer-create-tmp` (листинг 30.14) создается временный непечатаемый слой.

Листинг 30.14. Функция `ru-layer-create-tmp`

```
(defun ru-layer-create-tmp (/ all lay item new iter new-vla)
 (vlax-for item (setq lay (vla-get-layers (ru-obj-get-active-document)))
       (setq all (cons (vla-get-name item) all))
);_ end of vlax-for
 (setq iter 0)
 (while (vl-position
```

```
              (setq iter (1+ iter)
                    new  (strcat "RU-TMP-" (itoa iter))
              );_ end of setq
              all
         );_ end of vl-position
);_ end of while
 (setq new-vla (vla-add lay new))
 (vla-put-Plottable new-vla :vlax-false)
 (vla-put-Color new-vla 6)
    new-vla
);_ end of defun
```

Функция `ru-plot-layout-cfg` (листинг 30.15) получает в качестве аргументов объект компоновки и список параметров новой конфигурации печати. Функция устанавливает новую конфигурацию печати, а возвращает ранее действовавшую конфигурацию.

Листинг 30.15. Функция `ru-plot-layout-cfg`

```
(defun ru-plot-layout-cfg (layout cfg-lst / current)
 ;; Получаем текущую конфигурацию
 (setq current
  (mapcar (function (lambda (x) (vl-catch-all-apply x (list layout))))
   '(vla-get-configname vla-get-canonicalmedianame vla-get-centerplot
     vla-get-paperunits vla-get-plothidden vla-get-plotorigin
     vla-get-plotrotation vla-get-plottype vla-get-plotviewportborders
     vla-get-plotviewportsfirst vla-get-plotwithlineweights
     vla-get-plotwithplotstyles vla-get-stylesheet vla-get-viewtoplot)
 );_ end of mapcar
);_ end of setq
  (vla-refreshplotdeviceinfo layout)
 (mapcar
 (function (lambda (x y)
  (if (and y (not (vl-catch-all-error-p y)))
   (ru-obj-error-apply x (list layout y)))))
   '(vla-put-configname vla-put-canonicalmedianame vla-put-centerplot
     vla-put-paperunits vla-put-plothidden vla-put-plotorigin
     vla-put-plotrotation vla-put-plottype vla-put-plotviewportborders
     vla-put-plotviewportsfirst vla-put-plotwithlineweights
     vla-put-plotwithplotstyles vla-put-stylesheet vla-put-viewtoplot)
   cfg-lst
);_ end of mapcar
 current
)
```

Функция `ru-block-minsert-to-array` (листинг 30.16) получает в качестве аргумента примитив вставки блока и преобразует его в массив блоков, возвращая список элементов массива.

Листинг 30.16. Функция ru-block-minsert-to-array

```
(defun ru-block-minsert-to-array (block / param dxf)
  (setq param (ru-block-get-array-cfg block) dxf (entget block))
  (entdel block)
    (entmake
    (ru-ent-dxf-code-clear-list dxf '(-1 330 5 70 71 44 45) nil))
    (setq block (entlast))
    (if (< 1 (apply '* (car param)))
     (ru-obj-array-rectangular block (caar param) (cadar param) 1
        (caadr param) (cadadr param) 0)
     (list (vlax-ename->vla-object block))
   );_ end of if
)
```

Печать единичного листа осуществляется функцией ru-plot-unit-now (листинг 30.17).

Листинг 30.17. Функция ru-plot-unit-now

```
(defun ru-plot-unit-now (layout_obj device p_size unit plot_obj fname opt / do ins)
  (vla-highlight unit :vlax-true)
  (vla-update unit)
  (vla-setwindowtoplot layout_obj
    (setq ins (ru-obj-point-3d-to-2d (vla-get-insertionpoint unit)))
    (ru-obj-point-move ins p_size)
 );_ end of vla-setwindowtoplot
  (vla-highlight unit :vlax-true)
;|-------------------------------------------------------------------------
Подсвечиваем текущий лист и запрашиваем разрешение на печать. При этом даем
возможность стандартного предварительного просмотра
-------------------------------------------------------------------------|;
  (if (or (not opt) (ru-plot-ask-and-preview plot_obj))
    (progn
      (if fname
        (vla-plottofile plot_obj fname device)
        (vla-plottodevice plot_obj device)
     );_ end of if
   );_ end of progn
 );_ end of if
  (vla-put-visible unit :vlax-false)
  (vla-highlight unit :vlax-false)
)
```

Печать листа с опцией предпросмотра осуществляется функцией ru-plot-ask-and-preview (листинг 30.18).

Листинг 30.18. Функция ru-plot-ask-and-preview

```
(defun ru-plot-ask-and-preview (plot_obj / answer end)
;;; Запрос печати листа с опцией предпросмотра
 (while (not end)
```

```
    (setq answer (ru-get-kword "Печатать подсвеченный лист"
                   "Yes No Preview" "Yes"))
  (cond
   ((= answer "Preview") (vla-displayplotpreview plot_obj acfullpreview))
   (T (setq end T) (= answer "Yes"))
 );_ end of cond
);_ end of while
)
```

Преобразование трехмерной точки типа `Variant` в двухмерную осуществляется функцией `ru-obj-point-3d-to-2d` (листинг 30.19).

Листинг 30.19. Функция `ru-obj-point-3d-to-2d`

```
(defun ru-obj-point-3d-to-2d (point / tmp)
    (setq tmp (vlax-safearray->list (vlax-variant-value point)))
    (ru-conv-list-points-to-variant-array (list (car tmp) (cadr tmp)))
)
```

Для отлавливания ошибок в программе печати была разработана специальная функция `ru-obj-error-apply` (листинг 30.20), принимающая аргументы аналогично функции `vl-catch-all-apply` и возвращающая `T` в случае ошибки или `NIL` в противоположном случае. Кроме того, функция добавляет сообщение об ошибке в начало списка глобальной переменной `*ru-object-error-messages*`.

Листинг 30.20. Функция `ru-obj-error-apply`

```
(defun ru-obj-error-apply (func lst / err is-err)
 (setq err (vl-catch-all-apply func lst))
 (if (setq is-err (vl-catch-all-error-p err))
   (setq *ru-object-error-messages*
    (cons (substr (vl-catch-all-error-message err) 19)
       *ru-object-error-messages*))
);_ end of if
 is-err
);_ end of defun
```

В качестве "отходов производства" представим функцию, которую мы разработали, но не использовали в программе — она, может быть, пригодится читателям. Функция `ru-plot-get-plotters-papers` (листинг 30.21) возвращает очень длинный список всех конфигураций бумаги (листинг 30.22) для всех обнаруженных плоттеров. Первоначально мы хотели производить выбор конфигурации из генерируемого XML-файла, позволяющего выбрать конфигурацию бумаги из всех известных устройств и форматов, но в дальнейшем остановились на более простом варианте последовательного выбора из двух линейных списков (см. рис. 30.2 и 30.3).

Листинг 30.21. Функция `ru-plot-get-plotters-papers`

```
(defun ru-plot-get-plotters-papers (force / layout current res)
 (if (or force (not *ru-plot-plotters-cfg*))
```

```
  (progn
   (ru-splash-show "Ищу конфигурации бумаги для известных плоттеров")
     (setq *ru-object-error-messages* nil
         layout (vla-get-layout (ru-obj-get-model-space))
         current (ru-plot-layout-cfg layout
         '(nil nil nil nil nil nil nil nil nil nil nil nil nil nil))
       res (mapcar (function (lambda (x)
           (cond
            ((ru-obj-error-apply 'vla-put-configname (list layout x)))
            ((list x (ru-plot-get-layout-papers-cfg layout)))
             (t nil)))) (ru-plot-get-plotters-name))
     );_ end of setq
   (ru-plot-layout-cfg layout current)
   (ru-splash-hide)
   (setq *ru-plot-plotters-cfg*
        (vl-remove-if-not 'listp (vl-remove-if 'null res)))
 );_ end of progn
 *ru-plot-plotters-cfg*
);_ end of if
)
```

Листинг 30.22. Сокращенный пример списка всех конфигураций

```
(
 ("None"
   (
    ("700mm_(700.00_x_1000.00_MM)" "700mm (700.00 x 1000.00 MM)"
      1 (685.0 960.0))
    ...
    ("ANSI_A_(11.00_x_8.50_Inches)" "ANSI A (11.00 x 8.50 Inches)"
      1 (266.7 177.8))
     ("ARCH_C_(18.00_x_24.00_Inches)" "ARCH C (18.00 x 24.00 Inches)"
       1 (444.5 571.5))
     ("ISO_A0_(1189.00_x_841.00_MM)" "ISO A0 (1189.00 x 841.00 MM)"
       1 (1174.0 801.0))
     ("ISO_A4_(210.00_x_297.00_MM)" "ISO A4 (210.00 x 297.00 MM)"
       1 (195.0 257.0))
     ("ISO_A4_(297.00_x_210.00_MM)" "ISO A4 (297.00 x 210.00 MM)"
       1 (282.0 170.0))
     ...
     ("Sun_Hi-Res_(1280.00_x_1600.00_Pixels)"
       "Sun Hi-Res (1280.00 x 1600.00 Pixels)" 2 (1279 1599))
     ...
     ("Super_VGA_(600.00_x_800.00_Pixels)"
      "Super VGA (600.00 x 800.00 Pixels)" 2 (599 799))
     ...
     ("XGA_Hi-Res_(1200.00_x_1600.00_Pixels)"
       "XGA Hi-Res (1200.00 x 1600.00 Pixels)" 2 (1199 1599))
  )
)
```

```
  ("Autodesk DWF Writer"
    (
     ("User119" "ANSI A: 8.5 x 11 in" 1 (209.55 273.05))
     ...
     ("User124" "ISO A4: 210 x 297 mm" 1 (203.55 290.55))
     ...
     ("User126" "ISO A2: 420 x 594 mm" 1 (413.55 587.55))
     ...
     ("User140" "ARCH E3: 27 x 39 in" 1 (679.45 984.25))
   )
 )
  ...
  ("PublishToWeb JPG.pc3"
    (("Sun_Hi-Res_(1280.00_x_1600.00_Pixels)"
       "Sun Hi-Res (1280.00 x 1600.00 Pixels)" 2 (1280 1600))
     ...
      ("Super_VGA_(800.00_x_600.00_Pixels)"
      "Super VGA (800.00 x 600.00 Pixels)" 2 (800 600))
      ...
      ("UserDefinedRaster (1218.00 x 1576.00Pixels)"
        "User 1 (1218.00 x 1576.00 Pixels)" 2 (1218 1576))
      ...
   )
 )
)
```

Резюме

Мы посвятили целую главу программе, которая, как нам казалось, предназначена для ограниченной категории пользователей. Просто задача была нам самим интересна, т. к. потребовала детального изучения объекта печати и заставила разработать несколько интересных вспомогательных функций. Мы намеренно ушли от разработки большого диалогового окна — чтобы не терять времени.

Однако неожиданно программа оказалась очень нужной некоторым "теткам", располагающим приличными крупноформатными плоттерами. Привлекательными оказались следующие моменты:

- простой интерфейс — несколько последовательных щелчков мышью оказались удобнее разборки с достаточно сложным стандартным диалоговым окном;
- возможность визуального прицеливания массивом "конвертов" оказалась удобной для расположения будущего изображения на листе вместо выравнивания с помощью смещений;
- в частном случае, при достаточном размере бумаги программа позволяет вывести и один лист с точным соблюдением масштаба (для чертежей генплана, топографии и сетей это обязательное условие) или, по крайней мере, оценить возможность вывода на один лист (от печати-то всегда можно отказаться).

Разумеется, программа еще очень "сырая", не более чем Alpha-версия, требует ряда доработок, тестирования с различными устройствами печати (многие из которых имеют собственные "подлые" особенности) и версиями систем AutoCAD. Напрашивается сохранение любимой конфигурации для типовых условий работы и ее быстрое применение.

Мы привели это описание для того, чтобы показать, как это делается, т. е. процесс разработки, а не готовый результат.

ГЛАВА 31

Несколько примеров расчетных программ

Выполнение различных расчетов является одной из привлекательных черт систем автоматизированного проектирования. Действительно, в среде AutoCAD, обладающей такими превосходными средствами для программирования прикладных задач, можно рассчитать все что угодно. Однако использовать расчетные возможности нужно очень осторожно, т. к. выполнение неверных расчетов хуже, чем их отсутствие.

Программа, выполняющая расчет по неверной методике, опасна тем, что в результаты расчетов обычно не вникают: "машина умная, она рассчитала". А кто определит, какую методику считать "верной"? В официальных нормативных документах стараются избегать прямого изложения методик расчетов, а если такие методики и приводятся, то для случаев, основанных на фундаментальных законах природы. Методиками расчетов полны различные рекомендации, пособия, справочники проектировщиков. Доверять этим материалам можно, но осторожно. Авторы таких материалов не несут особой ответственности за конкретные решения и результаты использования методик. На любые рекомендации, выпущенные одним головным институтом, непременно найдутся критики из другого, вполне обоснованно указывающие на методические ошибки.

Мы разрабатываем строительную САПР, а расчетные ошибки в строительстве чреваты возможными авариями и гибелью людей. Но даже если проектные промахи не приводят к уголовному разбирательству, последствия могут быть неприятными, и не столько для проектировщиков, сколько для людей, которые будут жить и работать в зданиях, построенных по нашим проектам.

К счастью, большинство конструкторов-строителей, от которых в наибольшей степени зависит надежность зданий, являются людьми разумно консервативными. Они никогда не схватятся за программу, уверяющую, что "это все, что потребуется как чертежнику, так и проектировщику"— они знают, что такого не бывает, но охотно используют проверенные и хорошо зарекомендовавшие себя расчетные системы. К сожалению, пока таких систем, встроенных в систему AutoCAD, мы не встречали.

А как быть программисту, если на него наседают пользователи, требующие "сделать" расчетную программу? Ответ такой — надо очень хорошо изучить предметную область, не полагаясь на "постановку" задачи в стиле "надо сделать".

Кажется, что очень легко разработать, например, программу расчета теплопотерь помещений, основанную на подкрепленной авторитетом СНиП фундаментальной

формуле `Q = k*F*dT`. Площади ограждений помещений можно легко получить в системе AutoCAD, а коэффициенты теплопередачи — вводить или выбирать из справочников пользователя. Все сложности видятся только в организации удобного интерфейса. Такая программа выполняет арифметические вычисления и на вид безопасна. Однако ее применение может привести к совершенно неверным результатам, если по этой формуле будут рассчитываться теплопотери помещений в *многоэтажном* здании. Площади ограждений, их конструкции (т. е. коэффициент теплопередачи) и разность температур внутреннего и наружного воздуха могут быть одинаковы для комнат, расположенных на первом и двадцатом этажах. Одинаковыми будут и *теплопотери через ограждающие конструкции*. Но *расход тепла на отопление* одинаковых комнат будет разный, т. к. в расчетной формуле не учитывается расход тепла на нагрев инфильтрующегося в помещения наружного воздуха. Рассчитать объем поступающего воздуха уже не так просто, т. к. он зависит от многих факторов, в том числе от конструктивных особенностей конкретного здания.

Следовательно, такая программа и называться должна соответственно — не *"расчет расхода тепла"*, а *"расчет теплопотерь через ограждающие конструкции без учета инфильтрации"*, и пользователи должны знать пределы ее применения. Еще лучше, если уж браться за такую программу, то разработать ее с учетом всех возможных нюансов. Однако именно такие нюансы и могут превратить простую (в математическом отношении) программку в сложную комплексную систему.

Не случайно прикладные расчетные программы чаще всего разрабатывают или инженеры, "доквалифицировавшиеся[1]" до программистов, или программисты, очень хорошо изучившие предметную область.

Однако даже если разработчик очень хорошо знает предметную область, надо позаботиться, чтобы с программой было удобнее работать, чем без нее. Приведем пример собственной неудачной программы — пусть это будет упомянутый расчет теплопотерь. Эти расчеты проще всего выполнять в электронной таблице, но мы поддались на уговоры заказчика, которому хотелось делать эти расчеты в системе AutoCAD. В результате появилась программа, в которой можно рассчитать потери тепла через отдельное ограждение (стена, окно, пол, потолок) с возможным измерением размеров ограждений в рисунке (рис. 31.1).

Из расчетов отдельных ограждений, каждое из которых может быть сохранено и использоваться как аналог для других, можно сформировать данные о теплопотерях помещения (рис. 31.2). Помещения также можно использовать как прототипы для других объектов. Из потерь тепла отдельных помещений формируются потери тепла зданием (рис. 31.3). Программа полностью учитывает все особенности таких расчетов и прекрасно решает поставленную задачу.

Почему же мы считаем ее неудачной? Да потому, что работать с ней обычному пользователю неудобно. Традиционно такие расчеты выполняются в табличной форме. Расчет в электронной таблице (или в сетке базы данных) соответствует традиционной технологии и, как принято говорить, имеет интуитивно понятный интерфейс.

Заполнение характеристик ограждений и помещений в формах, написанных на языке DCL, непривычно. В такую форму нельзя вписать расчетную таблицу — слишком уж неуклюж язык описания диалоговых окон.

---

[1] Или "дисквалифицировавшиеся", так как программист-кодировщик, в западном понимании, часто рассматривается как чернорабочий.

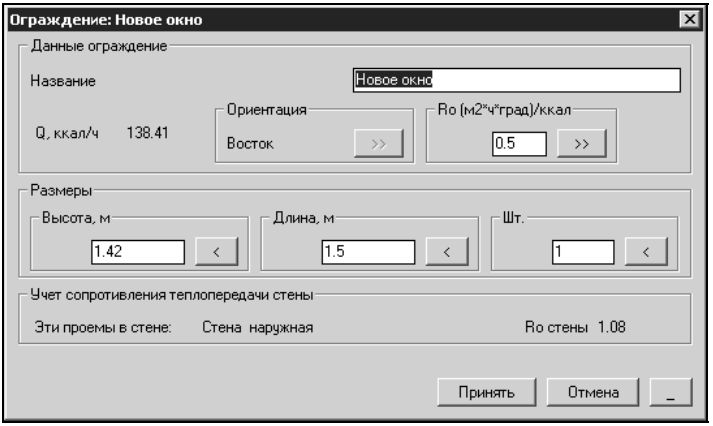


Рис. 31.1. Расчет теплопотерь одного ограждения

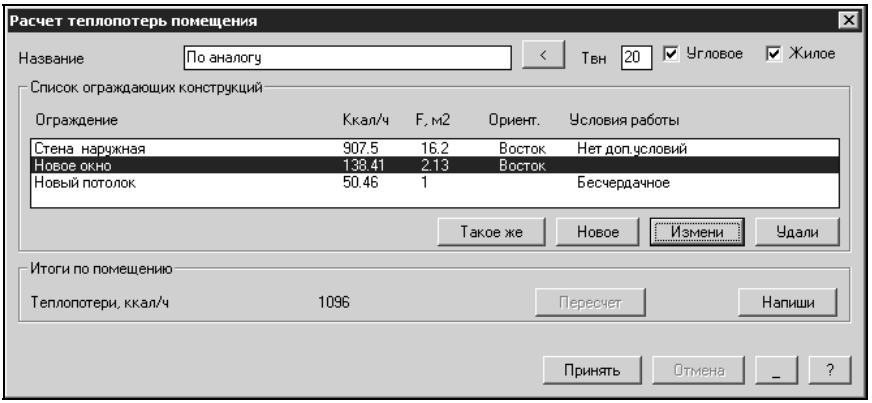


Рис. 31.2. Расчет теплопотерь помещения

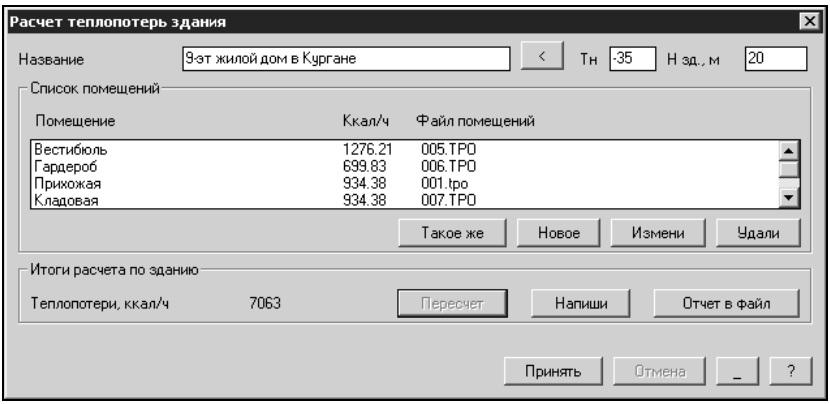


Рис. 31.3. Расчет теплопотерь здания

Единственным преимуществом такого варианта программы является возможность измерения размеров ограждений в системе AutoCAD, но за это небольшое и не решающее удобство приходится расплачиваться необходимостью щелчков по многочисленным кнопкам без полного обозрения всего расчета. Пользователю проще ввести число с клавиатуры, чем измерять его в системе AutoCAD. В результате эта программа попала в категорию "на любителя".

Расчеты объемов работ

При работе в системе AutoCAD чаще всего возникают идеи автоматического подсчета каких-то "количеств" с последующим использованием этих результатов. Реализовать такие идеи можно, и сделать это достаточно просто — мы приводили подобный пример в *главе 27*. Но надо хорошо подумать, стоит ли этим заниматься. Применительно к спецификациям оборудования этот вопрос рассмотрен в *главе 32*.

В этой же главе мы разберем несколько простейших программ для выполнения элементарных расчетов, которые могут пригодиться в виде компонентов для более сложных систем.

Измерение расстояний

Начнем с измерения расстояний. Это действие требуется выполнять часто, а стандартная команда DIST (ДИСТ) удобна только для разовых измерений. Попробуем разработать универсальную "рулетку", позволяющую:

- измерять расстояния от точки до точки;
- отменять измерение до неверно указанной точки;
- добавлять к измеренным расстояниям произвольные числа (например, существующих, но не доступных для измерения на плоском плане вертикальных участков), в том числе "прочитанные" надписи атрибутов, текстов и размеров;
- добавлять к измеренным "шаг за шагом" расстояниям дополнительные измерения "посторонних" участков и длин указанных примитивов;
- помечать измеренные участки цветом.

Как видите, запросы у нас скромные, и реализуем мы их в скромной функции (листинг 31.1).

Листинг 31.1. Функция `ru-trass-summ-length`

```
(defun ru-trass-summ-length (msg / list_point first_point next_point undo_count
add_len list_dist result ent lst_vert _add_dist)
;;; Локальная функция добавления очередного измерения
 (defun _add_dist (dist)
  (if (ru-is-point next_point)
  (setq first_point next_point) (setq next_point first_point))
  (setq list_dist
;; Добавляем измерение в список
   (cons dist list_dist)
```

```
;; Добавляем измерение к результату
          result     (+ result dist)
;; Добавляем очередную точку в список
          list_point
                     (cons next_point list_point)
;; Увеличиваем счетчик отката
          undo_count (1+ undo_count)
 )
)
;;; Главная функция
(setq undo_count 0 add_len 0 result 0.0)
(if
(setq first_point(ru-get-point-or-exit (strcat "Первая точка " msg) nil))
  (progn
   (setq list_point (cons first_point list_point)
         list_dist (cons 0.0 list_dist))
   (while (setq next_point (_ru-get-with-default
          (if (> (length list_point) 2) (strcat
            "Замерено сейчас... " (rtos (car list_dist) 2 3)
             " " (ru-unit-name) "."
            "\nВсего..... " (rtos result 2 3)
              " " (ru-unit-name) "."
            "\nСледующая точка " msg)
            (strcat "Следующая точка " msg))
            "Выход" 'getpoint nil
             (if (> undo_count 1) "Undo Sample Number Read"
                                  "Sample Number Read")
             (if (ru-is-point first_point)
                        (trans first_point 0 1) first_point)))
     (if (ru-is-point next_point)
        (setq next_point (trans next_point 1 0)))
     (cond
      ((= next_point "Undo")
       ;; Не отменяем первый сегмент
       (if (> undo_count 1)
         (progn
;; Стираем подсветку трассы
           (grdraw (trans (car list_point) 0 1)
             (trans (cadr list_point) 0 1) -1)
;; Делаем откат результатов
           (setq undo_count  (1- undo_count)
                 list_point  (cdr list_point)
                 first_point (car list_point)
                 next_point  (cadr list_point)
                 result      (- result (car list_dist))
                 list_dist   (cdr list_dist))
;; Снимаем подсветку с примитива-образца
           (if ent (progn (redraw  ent 4) (setq ent nil))))))
      ((= next_point "Sample")
```

```
;; Извлечение длины из указанного примитива
       (if (setq ent (ru-get-entsel-by-type
           (strcat "\nУкажите образец для '" msg "'")
             "Примитив недопустимого типа"
             (list "LINE" "LWPOLYLINE" "POLYLINE") nil))
           (progn
               (setq  ent (car ent)
                     lst_vert (ru-pline-list-vertex ent))
;; Подсвечиваем примитив, чтобы не забыть, что его сосчитали
               (redraw ent 3)
               (_add_dist (ru-geom-pline-length (car lst_vert)
                 (cadr lst_vert) T)))))
      ((= next_point "Number")
;; Ввод дополнительного числа - с клавиатуры или измерением
       (_add_dist (setq add_len (ru-get-dist
                 (strcat "Численная добавка для " msg) add_len nil))))
      ((= next_point "Read")
;; Чтение текстового числа
        (_add_dist (ru-get-calc-number (strcat
                 "Укажи размер, текст или атрибут с числом для " msg))))
       (T
;; Указана очередная точка
;; Подсвечиваем измеренный отрезок и добавляем измерение
        (grdraw (trans first_point 0 1) (trans next_point 0 1) 1)
          (_add_dist (distance first_point next_point)))
     )))) result)
```

Имея такую функцию, мы можем использовать ее в самых разных программах. Прежде всего, разработаем самую очевидную программу — "измерить и написать" (листинг 31.2).

Листинг 31.2. Файл ru_calc_sum_dist.lsp

```
(defun START (/ length_dist trass_name pt txt)
 (setq trass_name (ru-user-read-last-param "LastTrassName" "Трасса"))
 (while
  (setq trass_name (ru-get-string "Наименование измеряемой линии"
             trass_name))
  (if (not (zerop (setq length_dist (ru-trass-summ-length trass_name))))
    (progn
     (setq txt (rtos length_dist) pt  (ru-get-point-or-exit
               (strcat "\nТочка начала наименования '" trass_name "' ")
                     "Leader"))
     (cond
       ((= pt "Leader")
        (ru-draw-leader-and-two-string trass_name txt nil))
       ((ru-is-point pt)
        (ru-text-add trass_name pt (ru-normal-text-height) 0 nil)
        (if (setq pt (ru-get-point-or-exit
              (strcat "\nТочка начала текста длины '" txt "' ") nil))
              (ru-text-add txt pt (ru-normal-text-height) 0 nil)))))))
```

```
  (ru-user-write-last-param "LastTrassName" trass_name)
  (princ)
)
(START)
```

После измерения расстояния программа запрашивает точку для записи наименования трассы (ее можно указать и в графе таблицы) и точку для записи измеренного расстояния. При выборе опции **Leader** наименование и расстояние пишутся над полкой выноски.

Определение площадей

Расчеты площадей требуются очень часто — и для написания на планах, и для внесения в экспликации, и для прочих корыстных целей. Когда-то для определения площади использовалась команда AREA с последующим извлечением измеренной площади из системной переменной AREA, теперь мы можем воспользоваться объектными методами (листинг 31.3).

Листинг 31.3. Функция `ru-geom-area-ent`

```
(defun ru-geom-area-ent (ent)
 (ru-error-catch (function (lambda ()(vl-catch-all-apply
         'vlax-curve-getarea (list (vlax-ename->vla-object ent)))))
    nil))
```

Эта функция возвращает площадь примитива, но иногда требуется определить "теоретическую" площадь, заданную списком координат вершин. В системе AutoCAD проще всего по списку координат построить примитив, получить его площадь функцией `ru-geom-area-ent` и стереть примитив, но мы попробуем решить задачу путем вычислений. Расчет площади *выпуклого* многоугольника выполняется по формуле[1] 31.1.

$$S = \frac{1}{2}\left|\sum_{k-1}^{n}\left(X_k + X_{k+1}\right)\left(Y_k - Y_{k+1}\right)\right| \qquad (31.1)$$

Реализация соответствующей LISP-функции приведена в листинге 31.4.

Листинг 31.4. Функция `ru-geom-area-points`

```
(defun ru-geom-area-points (pnt_list)
 (abs (* (apply '+
             (mapcar
               (function
                  (lambda (p1 p2)
                    (* (+ (car p1) (car p2)) (- (cadr p1) (cadr p2)))))
                 pnt_list (cons (last pnt_list) pnt_list))) 0.5)))
```

[1] **algolist.manual.ru/maths/geom/polygon/area.php**.

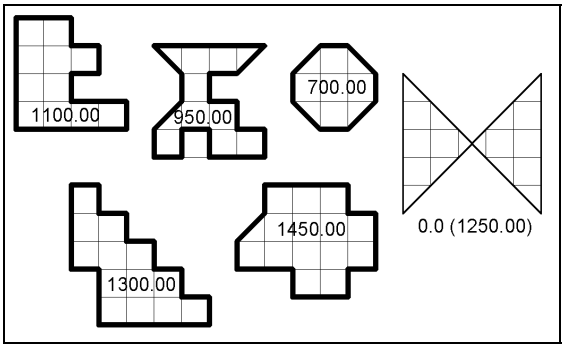


Рис. 31.4. Проверка расчета площадей

Многоугольник называется *выпуклым*, если никакая сторона многоугольника, будучи неограниченно продолженной, не разрезает многоугольник на две части. Однако наша функция правильно определяет площадь и не выпуклых, *практически встречающихся*, многоугольников (рис. 31.4), за исключением некоторых вариантов "бабочки".

Так как мы не можем утверждать, что функция `ru-geom-area-points` правильно определяет площадь во всех случаях, от ее применения мы воздержимся.

Спецкалькуляторы различного назначения

Располагая функциями для определения расстояний и площадей, мы можем разработать множество программ для различных прикладных вычислений.

На рис. 31.5 показано диалоговое окно *калькулятора площадей*. Программа предназначена для расчета любых площадей и написания результатов в рисунке. Наименование площади можно ввести в строке редактирования или в диалоговом окне функции `ru-get-string` с возможностью выбора из словаря. Площадь можно измерять несколько раз, добавляя или вычитая результат текущего измерения. Полученный результат можно написать (в таблицу, в контур, с выноской) в рисунке, при написании возможно задать подчеркивание наименования, результата и добавление единицы измерения площади. При щелчке по кнопке **Измерить** производится измерение площади контура.

Контур для определения площади создается одним из методов, выбираемых в диалоговом окне (рис. 31.6), выводимом функцией `ru-dlg-dcl-get-area`. Функция рисует замкнутую полилинию, определяет площадь полилинии с помощью функции `ru-geom-area-ent` и, если не установлен флажок **Рисовать контур**, стирает временный контур.

Приводить исходные тексты функций и диалоговых окон мы не будем, т. к. они объемны, но не содержат ничего интересного — рутинная работа с элементами диалоговых окон.

На рис. 31.7 показано диалоговое окно еще одного *спецкалькулятора ведомости изоляции трубопроводов*.

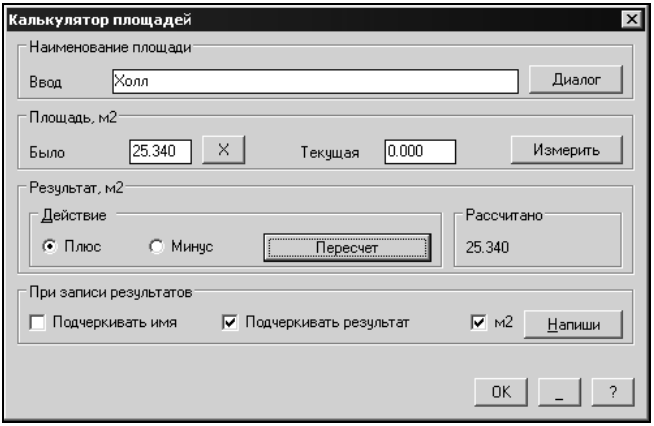


Рис. 31.5. Калькулятор площадей

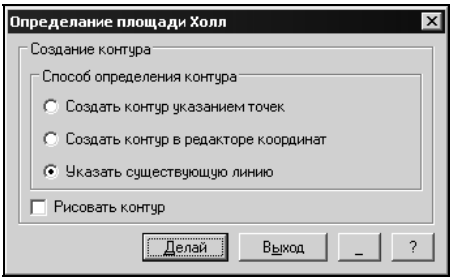


Рис. 31.6. Диалоговое окно создания контура

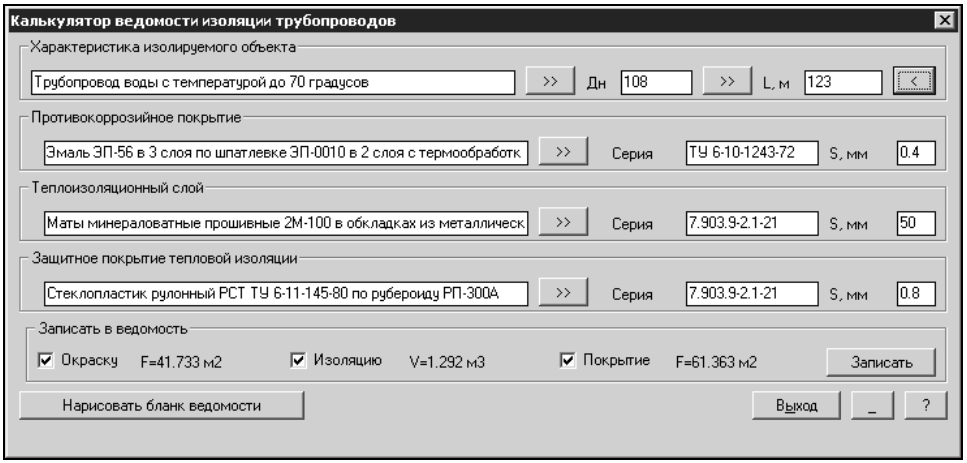


Рис. 31.7. Диалоговое окно калькулятора ведомости изоляционных конструкций

ВЕДОМОСТЬ ТЕПЛОВОЙ ИЗОЛЯЦИИ И ПРОТИВОКОРРОЗИЙНОЙ ЗАЩИТЫ

Наименование объекта	Характеристика объекта	Кол.	Противокоррозийная защита			Тепловая изоляция				Защитное покрытие
			Конструкция	Площадь, м2		Конструкция	Объем, м3		Чертеж	Конструкция
				ед.	Всего		ед.	Всего		
Трубопровод воды с	Дн=108.4	120 м	Покрытие	0.339	40.72	Маты из стеклянного	0.01	1.257	7.903.9-2.1-13	Стеклопластик
температурай до 70 градусов			масляно-битумное в 2			штапельного волокна				рулонный РСТ ТУ
Дн=108.0			слоя по грунту ГФ-021			МС-50 ГОСТ 10499-78				6-11-145-80 по
			S=0.2, ГОСТ 25129-82			S=50				рубероиду РП-300А
										S=0.8
Трубопровод воды с	Дн=273.4	34 м	Покрытие	0.858	29.16	Маты минераловатные	0.02	0.796	7.903.9-2.1-21	Листы из алюминиевых
температурай до 70 градусов			масляно-битумное в 2			прошивные 2М-100 В				сплавов ГОСТ 21631-76
Дн=273			слоя по грунту ГФ-021			обкладках из				S=1
			S=0.2, ГОСТ 25129-82			металлической сетки				
						ГОСТ 21880-86 S=50				

Рис. 31.8. Ведомость тепловой изоляции (фрагмент)

При проектировании технологических трубопроводов может применяться много разновидностей конструкций тепловой изоляции и противокоррозийной защиты. Этот калькулятор позволяет нарисовать бланк ведомости, рассчитывать площади поверхности трубопровода и защитного покрытия изоляции и объем теплоизоляционного слоя. Длина участков трубопроводов измеряется в рисунке, типовые конструкции изоляции выбираются из справочников. В результате может быть сформирована ведомость, показанная на рис. 31.8.

Разработаны и другие подобные калькуляторы, например, масс листовых и погонажных изделий, объемов насыпей и траншей.

Математика с числовыми текстами

Частенько пользователю приходится перемножать или суммировать числа, написанные в виде текста. Например, если требуется получить общий объем изоляционных конструкций, занесенных поэлементно в ведомость (см. рис. 31.8). Напишем функцию (листинг 31.5), позволяющую выполнять какие-то действия с числами, читаемыми в рисунке. Число может быть в виде текста, значения атрибута или размера. Функции передается один аргумент — строковое имя функции, которая должна применяться к двум числам. Обратите внимание на использование функций `eval` и `read` — именно эта пара позволяет обрабатывать аргумент, переданный в виде строки. Вычисление мы "оборачиваем" в традиционную функцию-ловушку `ru-error-catch`, чтобы предотвратить сбой при возможном делении на ноль или по иным заранее неведомым причинам.

Листинг 31.5. Функция `ru-text-eval`

```
(defun ru-text-eval (func / n1 n2 pt result txt_result)
 (while (setq n1 (ru-get-calc-number "Укажи первое число "))
  (if (/= n1 0)
   (progn
     (while (setq n2 (ru-get-calc-number "Укажи следующее число"))
```

```
        (setq result (ru-error-catch (function (lambda ()
                  ((eval (read func)) n1 n2))
              (function (lambda (x)
                (princ (strcat "Ошибка вычисления "
                         (vl-princ-to-string n1) func
                         (vl-princ-to-string n2) "\n")) n1)))
                txt_result (vl-princ-to-string result))
       (princ (strcat "\nРезультат: " (vl-princ-to-string n1) func
                 (vl-princ-to-string n2) "=" txt_result))
       (setq n1 result)
    )
     (setq pt (ru-get-point-or-exit
         (strcat "\nТочка начала текста результата '" txt_result "'")
         "Line"))
     (cond
;; Возможность написать текст вдоль линии
       ((= pt "Line") (ru-draw-txt-up-line txt_result))
       ((ru-is-point pt)
         (ru-text-add txt_result pt (ru-normal-text-height) 0 nil))
    ))))(princ))
```

Вызвать эту функцию можно из любого подходящего места в виде `(ru-text-eval "*")`, `(ru-text-eval "+")`, `(ru-text-eval "-")`, `(ru-text-eval "/")` или даже `(ru-text-eval "expt")`, хотя в практических целях применяется сложение или умножение. Так как в экзотических случаях может возвращаться не только численный результат, перевод результата в строку предусмотрен функцией `vl-princ-to-string`.

Быстрый прикидочный расчет диаметров трубопроводов и воздуховодов

Неопытных проектировщиков могут поставить в затруднительное положение самые простые задачи, например, определение диаметров трубопроводов или воздуховодов. Пока нет опыта, трудно сразу определить, какой диаметр требуется для пропуска такого-то расхода среды. Вспоминая сложные гидравлические расчеты, выполнявшиеся во время учебы, молодые специалисты с надеждой хватаются за программы, совмещающие черчение и расчеты.

В Советском Союзе было создано немало программ гидравлических расчетов различного назначения. Разрабатывали их весьма квалифицированные специалисты на высоком постановочном уровне. Неудобно только было применять эти замечательные программы из-за технологии прохождения расчетов (заполнение бланков, обработка в вычислительном центре, исправления, повторные расчеты). Технология была настолько неудачной, что большинство проектировщиков правдами и неправдами избегали машинных расчетов и продолжали работать традиционными методами. Сейчас положение дел иное — у каждого есть компьютер, имеются и хорошие программы, но беда в другом — современные расчетные программы бесплатно не распространяются. Вот и получается парадокс — опытная "тетка", работая на компьютере, держит под рукой таблицы или номограммы для гидравлического расчета, по которым, как и пятьдесят лет назад, *прикидывают* диаметры трубопроводов, ориентируясь на скорость среды и удельные потери давления.

Такая "методика" не так уж плоха — не всегда нужен, да и не всегда возможен настоящий гидравлический расчет с увязкой потерь давления, определением дроссельных диафрагм и прочими прелестями.

Для решения таких простых задач мы и включаем в нашу систему программу для *экспресс-проверки диаметров* трубопроводов и воздуховодов (рис. 31.9—31.11), позволяющую:

- выбрать вид системы;
- выполнить расчет или при заданном расходе, или при заданной скорости движения среды;
- выбрать проверяемый диаметр трубопровода или воздуховода;
- получить для выбранного диаметра скорость и удельные потери давления в выбранных единицах измерения;
- получить потери давления для введенной или измеренной длины участка;
- записать результат.

Программа всего лишь рассчитывает линейные потери по одному участку, но для опытной "тетки" этого бывает достаточно — она имеет замену всех таблиц. Диаметры именно задаются, а не рассчитываются (о расчете диаметров см. далее), т. е. выполняется *поверочный расчет*.

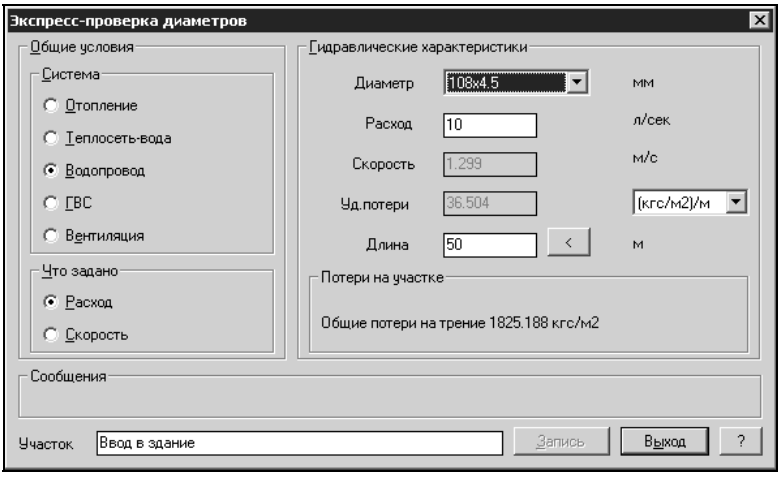


Рис. 31.9. Диалоговое окно проверки диаметров водопроводных труб

При проверке диаметров водопровода расход задается в л/с, как принято в таких системах, для других систем расходы задаются в кг/час или $м^3$/час.

Приводить полный текст программы мы не будем, разберем только фрагмент, по которому можно понять, как именно выполняется расчет потерь давления. Функция расчета потерь давления в трубопроводах системы отопления[1] (листинг 31.6) приве-

[1] Подобные функции используются и для расчета трубопроводов другого назначения.

дена нами не для того, чтобы можно было ею где-то еще воспользоваться. Попробуем разобрать принципы разработки функций в *расчетных* программах.

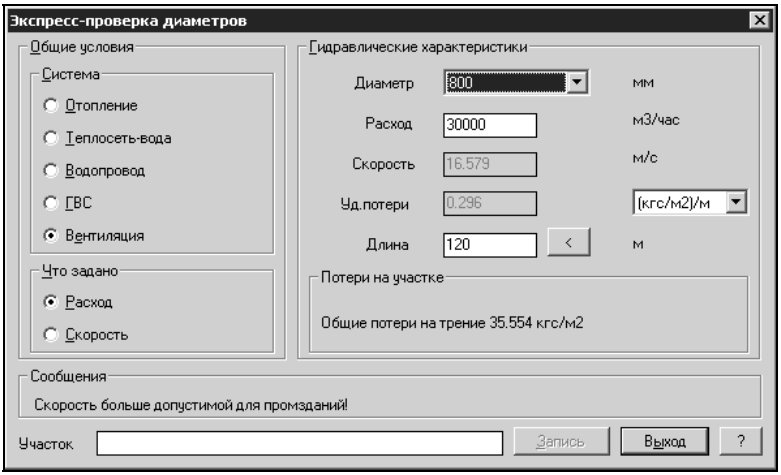


Рис. 31.10. Диалоговое окно проверки диаметров воздуховодов

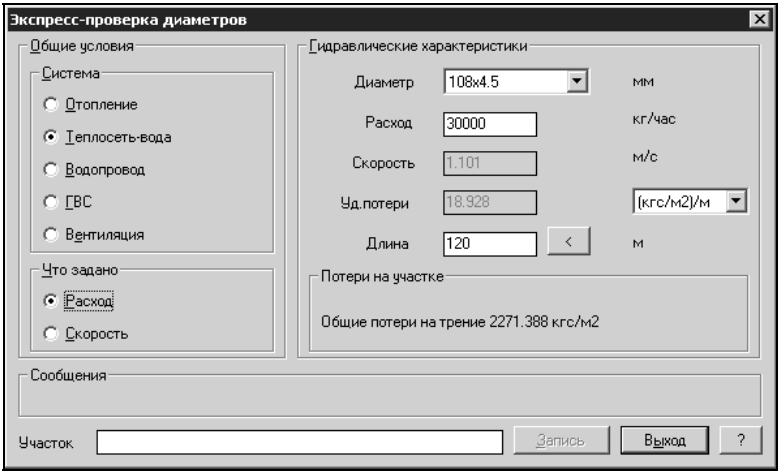


Рис. 31.11. Диалоговое окно проверки диаметров водяных тепловых сетей

Листинг 31.6. Функция `_eval_hw`

```
(defun _eval_hw (v g d ke si / density f)
;;; Расчет потерь давления в трубопроводе насосной системы отопления
;;; Аргументы:
;;; v - скорость воды, м/с, или NIL, если задан расход
;;; g - расход воды, кг/ч, или NIL, если задана скорость
```

```
;;; d - внутренний диаметр трубопровода, мм
;;; ke - эквивалентная шероховатость трубопровода, мм
;;; si - единицы измерения в системе СИ, иначе NIL
   (setq d       (* 0.001 d)
         density 983.0;; Плотность воды
         f       (/ (* pi (expt d 2)) 4));; Площадь сечения трубы
    (if g
;;; Если задан расход, определяется скорость
      (setq v (/ (/ (/ g density) 3600.0) f))
;;; иначе по скорости определяется расход
      (setq g (* (* (* v f) 3600.0) density))
   )
;;; Удельные потери на трение, кгс/м2
    (setq _specific_resistance
          (* (/ (/ 1.488 (expt (log (/ (* 5.5 d)
             (+ ke (/ (* 73 (expt d 2)) g)))) 2)) d)
             (/ (* (expt v 2) density) 19.62))
;;; пересчет единиц
    (if si (setq _specific_resistance (* _specific_resistance 9.81)))
    (setq _volume g _speed v);_ end of setq
    (set_tile "ed_volume" (rtos _volume 2 3))
    (set_tile "ed_speed" (rtos _speed 2 3))
    (set_tile "ed_pot" (rtos _specific_resistance 2 3))
    (set_tile "txt_result"
        (strcat "Общие потери на трение "
           (rtos (* _specific_resistance _length) 2 3) " " _name_lost)
   )
    _speed
)
```

Это локальная функция, вызываемая внутри основной программы, поэтому в ней используются несколько глобальных по отношению к этой переменной, но локальных в программе переменных (_specific_resistance, _volume, _speed, _length, _name_lost). Это не очень хорошо, но в программах, в которых применяется DCL, допускается. В данном случае не будем к этому придираться, хотя можно было результат вернуть списком, а установку значений полей редактирования диалогового окна выполнить в более подходящем месте программы.

Замечание

Внимательные читатели заметят, что в диалоговых окнах и функциях встречаются наименования единиц измерения, не соответствующие современным стандартам (килокалории, кгс/м^2). Связано это с тем, что в теплотехнической отрасли международная система единиц СИ[1] внедряется "со скрипом" из-за плохой наглядности внедряемых единиц измерения. Изымаемая из обращения килокалория (количество теплоты, требуемое для нагрева одного килограмма воды на один градус) была "интуитивно понятна", как понятны были единицы измерения давления (напора), выражаемые в метрах и миллиметрах водяного столба. Так как большинство проектировщиков по-прежнему предпочитают запрещенные единицы, в программах мы предусматриваем два варианта вывода результатов — на вкус пользователя.

[1] В СССР объявлена предпочтительной с 1963, а стандартной — с 1981 года.

Особенности программирования расчетов

Рассмотрим расчетную часть. В ней сделана попытка оптимизации функции в любимом LISP-программистами стиле — избавление от ненужных присваиваний и переменных. Заметим, что оптимизировать можно было бы и лучше, но в данном случае оптимизация просто вредна. В приведенной ранее функции `ru-geom-area-points` (см. листинг 31.4) также была проведена оптимизация, в результате функция стала и "красивой", и понятной. Но расчет потерь давления более сложен и стандартен. Его изучают студенты всех строительных специальностей (большинство "несантехников" об этом просто забыли), но в приведенной формуле с трудом разберутся даже те, кто занимается такими расчетами профессионально. Неизвестно, откуда взялись константы 1.488, 5.5, 73, можно догадаться только, что 19.62 — это 2g. Специалисты знают, что существует несколько вариантов определения коэффициента гидравлического сопротивления трению, но константы не похожи на используемые в известных методиках. Через некоторое время об этом забудет и автор программы.

А для кого мы пишем исходный текст программы? Не столько для "машины", сколько для человека, который с этой программой будет разбираться. Если в функциях общего назначения вполне уместно воспользоваться языковыми преимуществами LISP, избавиться от ненужных переменных, то в расчетных программах лучше не увлекаться оптимизацией, а написать исходный текст так, чтобы он прежде всего был понятен специалисту. В этом случае не повредят и лишние переменные, имена которым лучше назначать общепринятые в предметной области, например, не `specific_resistance`, а `H_l`[1].

Но и с именами переменных не так просто. Ученый люд очень любит придумывать сложные обозначения физических и технических величин — с верхними и нижними индексами, да еще составными. Возможно, это является мерой "крутизны" автора, но для программистов — беда. Обозначение нужно записывать в одну строку, но тогда общепринятое или даже стандартизированное обозначение будет непонятно никому, особенно после перевода на латиницу греческих символов. Мы предполагаем, что когда-то будет выработан и стандарт для "перевода" обозначений величин в компьютерный вид, но работать нужно сейчас. В книге мы старались давать именам переменных осмысленные английские имена, но для многих технических величин подобрать английский эквивалент просто невозможно, а опускаться до "транслита" ("rasxod_tepla") уже не хочется. Так как язык LISP позволяет использовать для имен переменных любые символы, рискнем сделать неожиданное предложение — в расчетных программах использовать русские имена переменных.

Попробуйте, например, придумать понятное английское имя переменной *"норма расхода горячей воды потребителем в час наибольшего водопотребления"*, учитывая, что в СНиП 2.04.01-85 эта величина обозначается трехэтажными символами (31.2).

$$q_{hr,u}^{tot} \tag{31.2}$$

[1] К сожалению, в некоторых СНиП введены такие буквенные обозначения для расчетных формул, которые просто невозможно записать в программах, поэтому приходится конструировать имена, хотя бы похожие на стандартные.

Может получиться нечто непроизносимое и непонятное наподобие **q_tot_hr_u**. Непонятное ни ученым, ни "англичанам". А на русском языке это будет **"норм_расх_гв_мах_час"**. Тоже неблагозвучно и некрасиво, но понятно. По крайней мере, "нашим" — а кто же еще будет читать наши исходные тексты программ.

Вернемся к определению потерь давления. Классическая формула 31.3 понятна любому инженеру, отражает физический смысл процесса, но не очень удобна как для ручных, так и для автоматизированных вычислений.

$$\Delta p = \left(\frac{\lambda}{d_в} l + \sum \zeta\right)\frac{\omega^2}{2}\rho \qquad (31.3)$$

где:

Δp — падение давления, вызванное трением и местными сопротивлениями, Па;

λ — коэффициент гидравлического трения;

$d_в$ — внутренний диаметр трубопровода, м;

l — длина участка сети, м;

$\sum \zeta$ — сумма коэффициентов местных сопротивлений на рассчитываемом участке;

ω — скорость жидкости в трубопроводе, м/с;

ρ — плотность жидкости в трубопроводе, кг/м^3.

Самым сложным является определение коэффициента гидравлического трения, для расчета которого существует несколько вариантов расчетных формул для различных режимов течения жидкости (ламинарный, переходный, турбулентный) и шероховатости труб. Авторы всех вариантов расчета коэффициента гидравлического трения (Мурин, Блазиус, Никурадзе, Киссин, Зусманович и др.) стремились добиться максимального соответствия расчетных и опытных данных, однако расчеты оставались достаточно сложными, т. к. в них необходимо учитывать плотность и вязкость жидкости, не являющимися постоянными. Поэтому для практического применения были разработаны формулы, предназначенные для типовых условий (температура воды 80 градусов, эквивалентная шероховатость труб 0.2, 0.5 и 1 мм), а по этим формулам были составлены таблицы и номограммы, использующиеся уже несколько десятилетий. Кроме того, для упрощения инженерных расчетов систем в целом были разработаны различные практические методики расчета (по характеристикам сопротивления, проводимостям, относительным расходам и т. п.) потерь давления в трубопроводах.

После появления первых ЭВМ, которые физически были большими, но имели мизерную, по нынешним меркам, память, в совершенствование расчетных формул включились программисты, старавшиеся использовать более удобные варианты расчета коэффициента гидравлического трения, применять функции, имеющиеся в системах команд конкретной системы, и преобразовывать расчетные зависимости для сокращения объема постоянных вычислений. В варианте, использованном нами в функции `_eval_hw` (см. листинг 31.6), коэффициент гидравлического трения определяется по формуле 31.4 Л. И. Кагана, преобразованной для воды с плотностью 983 кг/м^3 с заменой десятичного логарифма на натуральный.

$$\lambda = \frac{1.488}{\left(\ln \frac{5.5d}{K_э + 73\frac{d^2}{G}}\right)^2} \qquad (31.4)$$

Потери давления на трение (местные сопротивления можно учитывать в виде эквивалентной длины трубопровода) мы определяли по формуле 31.5.

$$H = \frac{\lambda}{d} l \frac{\omega^2}{19.62} \rho \qquad (31.5)$$

где:

H — потери давления на участке, кгс/м^2 (мм. вод. ст.);

ω — скорость жидкости, м/с.

Плотность воды, которую можно было бы оставить условно-постоянной, как это принято при расчете коэффициента гидравлического трения, мы оставляем в виде переменной. В функции `_eval_hw` (см. листинг 31.6), предназначенной для прикидочных расчетов, мы принимаем постоянную плотность воды 983 кг/м^3, но при выполнении более точных расчетов плотность воды мы будем определять функцией, приведенной в листинге 31.7.

Листинг 31.7. Функция `ru-eval-hot-water-density`

```
(defun ru-eval-hot-water-density (t_water)
;;; Плотность воды при заданной температуре
;;;  (ru-eval-hot-water-density 5.0) 999.91
;;;  (ru-eval-hot-water-density 65.0) 981.19
;;;  (ru-eval-hot-water-density 150.0) 910.3
;;; Использована формула Туркина В.П.
(- 1000.3  (* 0.06 t_water) (* 0.0036 t_water t_water))
)
```

Где брать формулы?

Большинство методик расчета в строительном проектировании использует данные, приводящиеся в виде таблиц или номограмм. Всегда возникает вопрос о том, где же взять формулы функций, по которым построены таблицы и графики. Все это изучается в курсе математики, но многие ли инженеры помнят, как это делается? Приведем некоторые практические рекомендации.

Во-первых, следует вести постоянную "охоту" за изданиями, в которых могут обнаружиться искомые формулы, а *во-вторых*, попытаться самостоятельно искомую формулу вывести.

Действуя по *первому варианту* (охота), мы, например, нашли формулу для определения плотности воды[1] (см. листинг 31.7).

[1] Имеется еще несколько вариантов этой формулы, выведенных разными авторами.

А вот для определения кинематической вязкости воды пришлось действовать по *второму варианту*. В этом случае лучше всего воспользоваться какой-либо программой аппроксимации функций. Можно работать с каким-нибудь "монстром", наподобие MathCAD, а можно разыскать простую программу, которая также справится с такой работой.

Для *точного* расчета коэффициента гидравлического трения, определения режима течения жидкости и выбора точной расчетной формулы требуется вычисление числа Рейнольдса. Число Рейнольдса определяют по формуле 31.6.

$$\mathrm{Re} = \frac{w_0 D_0}{\nu} \tag{31.6}$$

где:

w_0 — определяющая скорость потока, м/с;

D_0 — гидравлический диаметр трубы, м;

ν — кинематическая вязкость, м^2/с.

Кинематическую вязкость среды при ручных способах расчетов определяют по таблицам или принимают постоянной, пренебрегая ее зависимостью от температуры и давления. Формула 31.5 выведена для воды с плотностью 983 кг/м^3 с кинематической вязкостью 0.479×10^{-6} м^2/с. Такие условия примерно соответствуют обычным режимам работы систем и позволяют получать результаты с приемлемой для практических целей точностью. Выполняя расчеты на современных компьютерах, мы можем позволить себе и точные вычисления, особенно необходимые для многовариантных режимных расчетов.

Используя табличную зависимость кинематической вязкости воды от температуры и программу Approximator[1] (рис. 31.12), мы легко вывели расчетную формулу. Для этого достаточно заполнить таблицу (X — температура, Y — кинематическая вязкость), выбрать расчетную формулу, щелкнуть кнопку **Аппроксимация** и проанализировать результаты — отклонения вычисленных значений от заданных и вид графика. Разумеется, "математик" сразу, по виду графика, определит подходящую расчетную формулу, но и полуграмотный (в математическом отношении) инженер может просто перебрать все формулы и получить требуемый результат. При аппроксимации полиномом 4-й степени достигается наименьшее среднее отклонение на всем рассматриваемом диапазоне, но мы будем использовать более простую формулу 31.7, дающую отклонение до 10% только при температурах ниже 10°, т. е. вне применяемого диапазона температур воды.

$$\nu = \frac{33.30215068}{12.32052783 + t} \tag{31.7}$$

Теперь мы можем написать и LISP-функцию для определения кинематической вязкости воды (листинг 31.8). Самым трудным оказывается придумывание названия функции, а непременным условием — описание в комментариях источника информации и методики.

---

[1] **www.aproxim.narod.ru**.

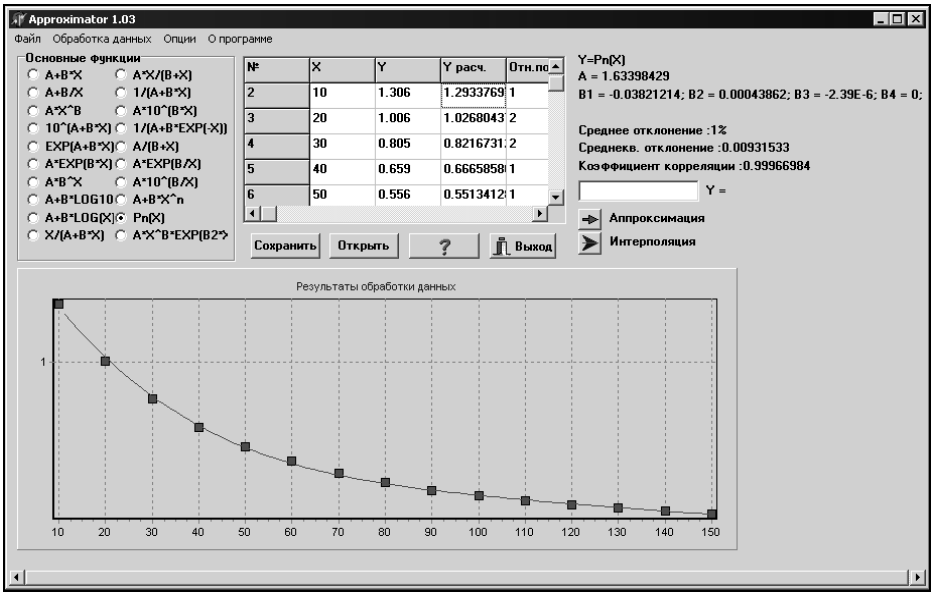


Рис. 31.12. Программа "Аппроксиматор"

Листинг 31.8. Функция `ru-eval-water-cinem-viscous`

```
(defun ru-eval-water-cinem-viscous (t_water)
;;; Кинематическая вязкость воды
;|
Получена аппроксимацией  таблицы 1-8 Справочника по гидравлическим сопротивлениям
И. Е. Идельчика издания 1960 года с использованием программы Approximator v.1.3.
t_water - температура воды, градусов Цельсия в диапазоне 10-150
Результат  - кинематическая вязкость*10^6 м^2/сек
|;
  (/ 33.30215068 (+ 12.32052783 t_water))
)
```

Алгоритм определения диаметров трубопроводов

Почти любая "тетка" на память знает, примерно какой диаметр трубопровода приблизительно соответствует заданному расходу среды. Программа этого знать не может и каждый раз, когда диаметр нужно рассчитать, должна определить его в первом приближении, а затем, возможно, произвести еще несколько итераций, добиваясь соблюдения нормативных ограничений. Беда в том, что ограничения имеют слишком большой диапазон. Минимальная скорость движения воды должна быть 0.25 м/с (для выноса пузырьков воздуха с потоком воды), а максимальная ограничивается условиями допустимого эквивалентного уровня шума и может составлять до 3 м/с в производственных зданиях. При таких условиях может быть очень много

расчетных итераций, и беда не в их количестве, а в том, что в результате могут быть сконструированы теоретически правильные, но практически неработоспособные системы.

Диаметры стояков систем отопления, как правило, принимаются конструктивно. СНиП рекомендует принимать потери давления в стояках не менее 70% общих потерь давления в циркуляционных кольцах, еще лучше, если потери в стояках будут составлять до 95% — тогда может быть обеспечена гидравлическая устойчивость системы. Но, если даже придерживаться рекомендаций СНиП и принимать минимальные диаметры стояков, остается простор для самодеятельности при расчете диаметров магистралей.

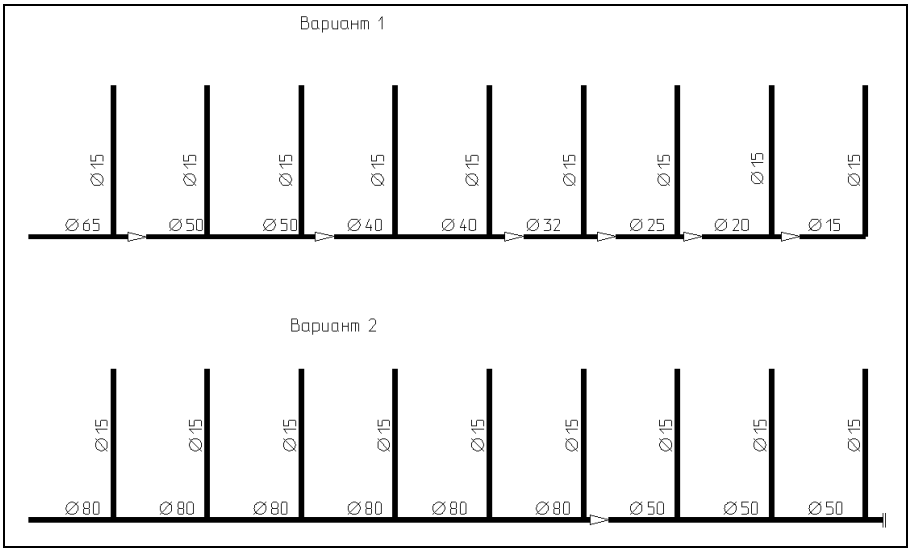


Рис. 31.13. Варианты диаметров магистрального трубопровода

На рис. 31.13 показаны два варианта диаметров магистралей при стояках с примерно одинаковой нагрузкой. *Первый вариант* часто получается в результате машинного расчета систем отопления. В нем соблюдаются и ограничения по скорости, и в магистралях теряется не более 30% располагаемого давления. Тем не менее, это очень плохое решение! Ориентируясь на минимальные металлоемкость и, как полагают некоторые проектировщики, стоимость, в проекте использовано семь типоразмеров труб. Однако подлинная экономичность системы оценивается совокупностью металлоемкости, затрат труда на монтаж, электроэнергии на перекачку теплоносителя. На сокращение затрат труда на монтаж (про которые проектировщики забывают — все, мол, заложено в сметы) значительно влияет индустриальность системы, которая может быть оценена в *коэффициентах индустриальности* и *унификации*, зависящих от количества типоразмеров труб. Не вдаваясь в доказательства, приводившиеся в литературе[1], скажем, что самым эффективным является *вариант 2*, в котором принима-

[1] Одельский Э. Х., Каган Л. И., Кирзнер Л. Х. Расчет систем центрального отопления и вентиляции на электронных вычислительных машинах. — Минск, 1974.

ется постоянный диаметр, равный диаметру для всей ветки, пока нагрузка не снизится до 0.33 от нагрузки всей ветви, а далее также принимается постоянный диаметр до конца ветки. Совсем немного уступает по показателям так любимый практиками вариант с постоянным диаметром магистрали, и на последнем месте находится *вариант 1.*

Расчетные формулы приведем сразу в виде функций (листинги 31.9—31.12). Во всех функциях аргументом является расход теплоносителя в т/ч (тонны в час), а результатом — оптимальный внутренний диаметр в метрах. В вызывающей программе принимается ближайший больший диаметр из стандартного типоразмерного ряда.

Листинг 31.9. Функция `ru-eval-diam-hw-st`

```
(defun ru-eval-diam-hw-st (g)
;;; Оптимальный внутренний диаметр стояка отопления
  (* 0.036 (expt g 0.53)))
```

Листинг 31.10. Функция `ru-eval-diam-hw-mag-33`

```
(defun ru-eval-diam-hw-mag-33 (g)
;;; Оптимальный внутренний диаметр конечных участков ветки
;;; с 1/3 нагрузки
 (* 0.022 (expt g 0.49)))
```

Листинг 31.11. Функция `ru-eval-diam-hw-mag-100`

```
(defun ru-eval-diam-hw-mag-100 (g)
;;; Оптимальный внутренний диаметр начальных участков ветки
  (* 0.034 (expt g 0.49)))
```

Листинг 31.12. Функция `ru-eval-diam-hw-mag-common`

```
(defun ru-eval-diam-hw-mag-common (g)
;;; Оптимальный внутренний диаметр сборных магистралей
  (* 0.037 (expt g 0.49))
)
```

Резюме

Для чего мы в очередной раз отвлеклись на рассуждения, не относящиеся непосредственно к программированию для системы AutoCAD? В начале главы мы сделали "грязный" намек на якобы неправильные методики расчетов, а сейчас привели конкретный пример. Неверность методики заключается не в том, что в ней использованы "неправильные" формулы — такого практически не бывает, а в том, что при формальной правильности всех расчетных компонентов может получиться принципиально неверный результат. В примере с диаметрами магистралей опасность не в том, что в первом варианте хуже показатели унификации, а в том, что такая система еще и гидравлически менее устойчива и, при отклонении располагаемого напора от расчетного значения, будет происходить и тепловая разрегулировка здания. Регули-

ровке, надежности и работе в нестационарных режимах при изучении курса отопления уделяется недостаточно внимания, а использовать машинные расчеты рационально именно для таких режимов.

Мы специально привели ссылку на столь "древний" труд Эммануила Хацкелевича Одельского и его коллег, в котором, наряду с давно устаревшими сведениями по заполнению ячеек памяти ЭВМ "Проминь", содержатся и вечно актуальные алгоритмы. Если программист берется за разработку подобных задач, то он просто обязан изучить все, что было написано по теме работы. Вообще большинство научно-технических изданий выпущено в лучшем случае в конце 80-х годов прошлого века, а в новой России, при огромной номенклатуре компьютерной литературы, не находится места для изданий, систематизирующих и излагающих расчетные методики на современном уровне.

ГЛАВА 32

Спецификации оборудования

В этой главе будет очень много рассуждений и очень мало "исходников". Большая часть текста может быть приравнена к постановке задачи. Реализация задачи достаточно проста и многие ее части мы уже выполнили в предыдущих главах. Программирование подготовки спецификаций более связано с разработкой приложений баз данных, не являющимися темой книги, чем с системой AutoCAD. Более важным нам показалось отразить то, что не написано ни в одной книге по программированию — особенности работы с этими важнейшими документами. Важнейшими потому, что на основании спецификаций составляются сметы, а многие виды работ можно выполнять только по смете, вообще не имея никаких чертежей.

Что такое *спецификации* — знает любой проектировщик[1]. В последние годы в проектной документации для строительства применяются спецификации оборудования, изделий и материалов по ГОСТ 21.110-95 СПДС и спецификации к планам расположения коммуникаций, конструкций, чертежам установок систем, выполняемые по ГОСТ 21.101.93.

За последние 30 лет спецификации делались разными способами:

- "угловые" — кому как вздумается;
- по форме 1 ГОСТ 21.104;
- сводные спецификации на заглавном листе;
- заказные спецификации;
- заявочные ведомости;
- ведомости объемов строительных и монтажных работ;
- ведомости потребности в материалах;
- спецификации оборудования.

Многообразие форм документов было вызвано существовавшей в СССР распределительной системой. Часть спецификаций, входивших в состав проектной документации, была предназначена для понятных задач — определения необходимых ресурсов для строительства объекта, установления его сметной стоимости и производства

[1] Материалы этой главы не распространяются на спецификации, составляемые по стандартам ЕСКД.

строительно-монтажных работ. Всякая анархия и самодеятельность в этой части была прекращена после ввода в действие стандартов СПДС.

Вторая часть спецификаций была предназначена для решения неразрешимой задачи — обеспечения объектов строительства оборудованием, изделиями и материалами. Существовавшая в СССР планово-распределительная система, при всех ее достоинствах, не позволяла заказчику просто купить необходимое оборудование и материалы — все надо было "выбивать". Практически все позиции были дефицитными. Промышленность, за исключением некоторых отраслей, была ориентирована на "вал", заводы выпускали то, что было тяжелее по весу и попроще в изготовлении — чугунные котлы, задвижки и радиаторы, толстостенные стальные трубы. Многое сантехническое и вентиляционное оборудование изготавливалось в известных учреждениях с соответствующим качеством. Капитальное строительство развивалось высокими темпами, а производство оборудования всегда отставало. В результате по стране мотались "орды" снабженцев, занимавшихся "доставанием" оборудования. Государство должно было от этих "орд" отбиваться и создавало "оборонительные линии" в виде различных "Главкомплектов", в которые нужно было подавать "заявочные документы", а потом "защищать" их. Тогда и начали плодиться различные формы документов, положения о порядке обеспечения материалами, опросные листы, перечни дефицитного оборудования и различной *обосновывающей* документации — все для того, чтобы под благовидными предлогами не давать. В ответ снабженцы стали ездить на "защиту" целыми "отрядами", с машинистками, пишущими машинками, запасами бумаги со всеми видами печатей, грузовиками с проектной документацией. Так как все заявки урезались, то заявляли много ненужного, составляя для этого фиктивные документы и проекты.

Для того чтобы упорядочить планирование производства и распределение, намечалась и *механизация* этой увлекательной симуляции деятельности. Были введены коды предприятий, изделий, материалов и видов работ, предусмотрены, якобы предназначенные для машинной обработки, графы заказных спецификаций — фактически этими кодами просто стращали снабженцев, да нагружали ненужной работой проектировщиков. Предполагалось, что в светлом будущем все это будет использоваться в неких единых системах механизированной обработки информации.

На память о тех временах у нас сохранилась форма "Проект плана материально-технического снабжения на 1980 год" с грозными указаниями по заполнению. Эту форму мы использовали для извлечения кодов скудной номенклатуры выпускаемого оборудования — применять можно было только то, что указано в проекте плана.

> Вершиной творения воспаленного воображения идеологов механизации стал, пожалуй, ГОСТ 21.204-81 "Паспорта строительных рабочих чертежей зданий и сооружений", в соответствии с которым на все рабочие чертежи, кроме объектов жилищно-гражданского строительства, проектная организация должна была выполнять паспорт по жуткой форме и направлять два экземпляра копий на перфокартах в ЦНИИПромзданий. Безусловно, паспорта рабочих чертежей объектов были нужны, и остались нужны сейчас, но шесть форм с несколькими сотнями ячеек, их подготовка и обработка "с помощью селекторов (универсального типа с поворотной площадкой ударного или вибрационного действия) и сортировальных спиц" поразили даже бывалых бюрократов. К счастью, этот стандарт приказал долго жить, не начав реально действовать. *(Сергей Зуев)*

Многие здравые идеи были погублены "топорным" исполнением. Например, после появления указаний о необходимости выполнения в составе рабочей документации

ведомостей потребности в материалах, строители-практики решили, что наконец-то Госстрой СССР возложит на проектные организации чрезвычайно трудоемкий подсчет количества материалов "до гвоздя". Вскоре выяснилось, что это очередной трюк "механизаторов", только теперь из НИИ экономики строительства. "Установленная номенклатура" материалов оказалась чрезвычайно укрупненной, например "прокат толстолистовой", и нужной только для сбора статистики в масштабах страны. До сбора и обработки ведомостей потребности в материалах дело так и не дошло, но лет десять проектировщики переводили тонны хорошей бумаги на практически никому не нужные бумажки.

После перехода к рыночной экономике все проблемы с обосновывающей документацией исчезли. Остались одни обосновывающие бумаги — деньги. Эти бумаги также стали дефицитными, ненужного оборудования никто приобретать не будет, но всем необходимым, с наилучшим соотношением цена-качество, стройку укомплектовать требуется и сейчас. Для этого и предназначены спецификации оборудования, необходимость в автоматизации составления которых не вызывает сомнений.

Автоматизация работы большинства проектных организаций начиналась с составления сметной документации. При всех несовершенствах использовавшихся тогда технологий выпуск смет с использованием ЭВМ давал огромный реальный эффект. Конечно, уже в 70-е годы прошлого века выполнялись и инженерные расчеты, но главным реальным продуктом, который мог "подержать в руках" заказчик, были все-таки сметы. После появления стандарта СПДС на спецификации оборудования разработчики некоторых систем автоматизированного выпуска смет предусмотрели выпуск и спецификаций, но все было поставлено "с ног на голову" — спецификация составлялась на основании сметы, а не на оборот. Проектировщики должны были каким-то образом задать номенклатуру и объемы работ, т. е. практически написать спецификации, а только потом, после составления сметы, получить отпечатанную спецификацию оборудования. При этом спецификация соответствовала стандарту только по форме, а наименования изделий и единицы измерения выводились в формулировке сметных норм. Такие спецификации не были пригодны для работы. Сметные системы хорошо работали с большими базами данных, и была возможность вести и настоящие базы оборудования и выпускать настоящие спецификации, но многоступенчатый процесс "бланкового" ввода данных делал это невыгодным. Быстрее было написать спецификации вручную.

> После появления в нашем институте мини-ЭВМ СМ-1420 с аж шестью терминалами, за которыми могли работать обычные инженеры, мы первым делом (вернее, вторым, после традиционных смет) автоматизировали выпуск спецификаций. Замечательный программист Сергей Николаевич Жаворонков по моему "наущению" тайком, в течение месяца разработал систему, позволяющую обычному, как бы теперь сказали, пользователю, вести банк данных оборудования и одновременно сразу выпускать спецификации. После формирования спецификации на первый реальный объект данные по примененному в нем оборудованию можно было использовать во всех последующих спецификациях. При этом не нужно было заносить оборудование впрок, а любую имеющуюся запись можно было использовать в качестве шаблона для другой. Тогда это казалось чудом. А тайком приходилось делать потому, что машинного времени и шести терминалов не хватило бы на весь институт. Пока другие отделы заполняли бланки для отмирающих ЕС ЭВМ, наш уже выпускал реальную продукцию. *(Сергей Зуев)*

После появления и массового распространения персональных компьютеров появились, но, к сожалению, очень неэффективно используются, новые возможности по автоматизации составления спецификаций.

Этот объемный экскурс в недавнюю историю мы написали для того, чтобы молодые разработчики знали, от чего мы избавились, и что потеряли. Знать историю необходимо хотя бы для того, чтобы не повторять прошлых ошибок. Люди, о которых мы писали с иронией, тоже искренне хотели делать "как лучше".

Что такое спецификация оборудования

Для того чтобы понять, как автоматизировать выпуск спецификаций оборудования, разберемся, что мы должны получить на выходе.

Спецификация оборудования, изделий и материалов, выполняемая по ГОСТ 21.110-95 СПДС (далее будем именовать Спецификация) — текстовый документ, определяющий состав оборудования, установок, изделий, устройств и материалов, предусмотренных рабочими чертежами соответствующего основного комплекта.

Документ по существу является сводной спецификацией к соответствующему комплекту рабочих чертежей и предназначен для комплектования, подготовки и осуществления строительства, а также для составления сметной документации ресурсным (ресурсно-индексным) методом.

ГОСТ 21.110-95 и дополняющие его "Методические рекомендации по составлению спецификации оборудования, изделий и материалов МР 21.01-95" вызывают противоречивые чувства. С одной стороны, это очень полезные документы, четко установившие роль спецификаций оборудования и, наконец-то, отменившие "мертвые" стандарты на ведомости объемов строительных и монтажных работ и ведомости потребности в материалах. С другой стороны, стандарт (как и многие другие стандарты СПДС) совершенно не учитывает современных информационных технологий, их реальных возможностей и реальных ограничений. Не учитываются и современные формы организации строительного производства и новые взаимоотношения между заказчиком (в современном понимании — инвесторами), проектировщиками и строителями.

Начнем критику с последнего утверждения. Несколько десятилетий фактическим инвестором в строительство было государство, устанавливающее "правила игры". Проектировщикам жестко предписывалось, как и в каком объеме выпускать документацию. Стоимость проектных работ диктовалась государством с учетом предписанной номенклатуры и детализации. Многие разделы документации нуждались в последующей доработке силами подрядных организаций — составление монтажных и деталировочных чертежей, определение реальной потребности в материалах и изделиях. Строители не были в обиде — затраты на эти работы входили в нормативы накладных расходов. Заказчику предписывалось укомплектовать строительство "оборудованием, поставляемым заказчиком". Как это делалось, мы писали в начале главы. Строительство подрядным способом велось, в основном, достаточно крупных объектов. Такие работы, как перепланировка и переоборудование квартир или строительство коттеджей, да еще по проектам, вообще не выполнялись. К сожалению, ГОСТ 21.110-95 написан так, будто бы жизнь не изменилась.

Представьте, что вы законопослушный частный предприниматель, приобрели квартиру для переоборудования под магазин (типичная ситуация), в соответствии с нормативными требованиями заказали проект перепланировки (за солидные деньги),

проект прошел государственную экспертизу, вы закупили по Спецификации все материалы и изделия и наняли бригаду рабочих. Но вдруг выясняется, что, например, в чертежах водопровода и канализации, предусматривающих монтаж системы из современных металлополимерных труб, заложено лишних десять метров "погонажа", но нет ни одной соединительной детали и крепления, стоимость которых весьма значительная. Деньги истрачены на ненужные трубы, а смонтировать их нельзя. Потом аналогичная ситуация будет с канализацией, отоплением и вентиляцией. Проектировщики в ответ на претензии покажут ГОСТ, где написано "элементы трубопровода (отводы, переходы, тройники, крестовины, фланцы, болты, гайки, шайбы, прокладки) в Спецификацию не включают", и любая экспертиза их поддержит (разве что заставит исключить лишние трубы, да еще с учетом места, занимаемого арматурой — тогда вы вообще не сможете списать затраты). А если количество таких позиций исчисляется тысячами штук?

Конечно, толковые проектировщики, дорожащие заказами, да еще выполняющие проектные работы в частном порядке, проигнорируют неразумный стандарт и сделают все, как надо. Но и на них может "наехать" орган экспертизы, указать на нарушение стандарта и даже, под благовидным предлогом, а на самом деле — для устранения конкурента "дружественной" проектной организации, лишить лицензии на строительную деятельность.

В подобной ситуации могут оказаться и проектно-строительные фирмы, осуществляющие нормальное современное проектирование и строительство "под ключ".

Теперь поговорим о несоответствии компьютерным технологиям. Форма Спецификации содержит девять граф:

- Графа 1 (Позиция) — позиционное обозначение (марка) оборудования, предусмотренное рабочими чертежами соответствующего комплекта.
- Графа 2 (Наименование и техническая характеристика) — наименование и техническая характеристика оборудования в соответствии с требованиями государственных стандартов и технических условий.
- Графа 3 (Тип, марка, обозначение документа, опросного листа) — тип, марка оборудования, обозначение стандарта, технических условий, а также (при необходимости) обозначение опросного листа.
- Графа 4 (Код оборудования, изделия, материала) — код оборудования по Общероссийскому классификатору продукции (ОКП).
- Графа 5 (Завод-изготовитель) — наименование завода-изготовителя для серийно изготавливаемого отечественного оборудования (для импортного оборудования — страна и фирма).
- Графа 6 (Единица измерения) — обозначение единицы измерения.
- Графа 7 (Количество).
- Графа 8 (Масса единицы) — масса единицы оборудования в килограммах.
- Графа 9 (Примечание) — дополнительные сведения.

В целом структура формы Спецификации пригодна для создания баз данных, в которых можно хранить как каталожную информацию, так и готовые Спецификации по конкретным объектам. Однако указания по заполнению граф (а особенно приме-

ры составления Спецификаций) убеждают, что разработчики нормативного документа имеют смутные представления об информационных технологиях и сделали все, чтобы усложнить автоматизированную подготовку спецификаций.

Введя понятную по названию графу **Позиция**, стандарт начинает запутывать проектировщиков, указывая, что в этой графе должны записываться *позиционные обозначения* (марки), а для изделий, не имеющих этого самого позиционного обозначения, уже в графе 2, перед их наименованием, следует указывать *порядковый номер* их записи в Спецификацию, да еще в пределах раздела. Вот на такие "крючки" и "ловили" снабженцев в "приснопамятные" времена! Вручную так писать можно, а формировать спецификацию автоматически — очень сложно. А ведь можно было сформулировать, что в графе 1 указывают "марки, порядковые номера и иные обозначения, предусмотренные рабочими чертежами". Да еще набраться смелости и добавить: "При соответствии позиций изделий в рабочих чертежах спецификации оборудования допускается не выполнять на листах рабочих чертежей спецификации установок и систем, предусмотренные соответствующими стандартами СПДС".

Разъяснения по графе 2 еще более усугубляют положение. Многие стандарты предусматривают "многоэтажное" обозначение материалов и изделий, которое даже в текстовом процессоре MS Word трудно изобразить без подключения редактора формул или хитрого форматирования таблиц, например:

$$\text{Полоса}\ \frac{5 \times 50\text{-В-2 ГОСТ 103-76}}{\text{Ст3пс1-II ГОСТ 535-88}}$$

Много лет проектировщики мучились, проклиная авторов таких изобретений и вписывая замысловатые конструкции в строки бланков спецификаций.

В базе данных наименование изделия будет храниться в одном поле в виде строки, обозначение — в другом поле. Заполнять автоматически тысячи подобных граф (да еще во множестве вариантов) просто невозможно. Недаром даже во включенных в МР 21.01-95 примерах составления Спецификаций нет ни одного, в котором изделие было бы записано в соответствии с текстом самого документа.

Проектировщики, при необходимости указать полную техническую характеристику, часто[1] просто пишут так, как показано в табл. 32.1. Такая запись облегчает и производство работ, т. к. на стройке нет никакой возможности иметь все стандарты для расшифровки замысловатых обозначений.

Таблица 32.1. *Пример 1*

Наименование и техническая характеристика	**Тип, марка, обозначение документа, опросного листа**
Полоса стальная горячекатаная толщиной 5 мм, шириной 50 мм, обычной точностью прокатки, с серповидностью по классу 2 из стали марки Ст3пс, категория проката 1, группа проката 2	ГОСТ 535-88

[1] Еще чаще пишут просто "Полоса 5 × 50", хотя мы бы рекомендовали всегда указывать подробные технические характеристики по принципу — "что стандарт зашифровывает, мы раскрываем". Не жалейте компьютеры, пожалейте прорабов!

Писать вручную множество раз такие длинные формулировки сложно, но можно воспользоваться предусмотренными стандартами способами сокращений повторяющихся в графе текстов. При использовании базы данных любой текст вручную набивается только один раз, при этом можно применять любую существующую запись в качестве прототипа.

Еще одним образцом нетехнологичных заполнений граф является запись вентиляторов, имеющих различные варианты исполнений и комплектаций электродвигателями. В примерах составления Спецификаций имеются формулировки, одна из которых показана в табл. 32.2.

Таблица 32.2. *Пример 2*

Позиция	Наименование и техническая характеристика	Тип, марка, обозначение документа, опросного листа
	2. Вентилятор радиальный, исполнение 1,	ВР-Ц4-75.1-2,5-Л.05
	диаметр колеса 1.1 Дном, положение Л 0	4АА56А4
	с электродвигателем 1375 об/мин, 0.12 кВт	ТУ 22-59-33-85

В этом примере два фрагмента полного наименования изделия (обозначение вентилятора и марка электродвигателя) перенесены в графу 3, перед наименованием, при пустой графе 1, проставлен пресловутый порядковый номер (при том, что именно вентиляторы обычно имеют позиционное обозначение, но уже в спецификации к установке системы). Единое наименование разбито на несколько строк, при этом обычно марку электродвигателя стараются написать в той же строке, где приводятся его характеристики. Корни таких примеров растут из "древних" Рекомендаций ГПИ Сантехпроект, еще из "докомпьютерной эры". Похоже, что кое для кого время остановилось на месте. Разве трудно сообразить, что записывать такие изделия нужно примерно так, как показано в табл. 32.3 (заодно избавившись от спецификаций установок на листах).

Таблица 32.3. *Пример 3*

Позиция	Наименование и техническая характеристика	Тип, марка, обозначение документа, опросного листа
В1.2	Вентилятор радиальный ВР-Ц4-75.1-2,5-Л.05, исполнение 1, диаметр колеса 1.1 Дном, положение Л 0 с электродвигателем 4АА56А4 1375 об/мин, 0.12 кВт	ВР-Ц4-75.1-2,5-Л.05

Продолжим критику. Коды оборудования, заносимые в графу 4, были придуманы именно для механизированной обработки. По существу эти коды должны были бы быть уникальными индексами, так необходимыми при обработке данных. Стесняясь признать тот факт, что конечная цель кодирования провалена, стандарт дает дели-

катную оговорку, что если "отсутствует информация по кодам ОКП, необходимая для заполнения графы 4, то эту графу не заполняют". Вот за это спасибо! Ушлые проектировщики, очень любящие всякие "если, нельзя, но очень хочется", "допускается", "если нужно, но не хочется", давно перестали заполнять эту графу, понимая ее бесполезность для практических целей. Продолжают заполнять те, кто по инерции "идет строем". Пусть бы была эта графа "про запас", но она занимает место, необходимое для расширения графы 3, т. к. существует множество обозначений документов, не вписывающихся в ширину этой графы, а правильный автоматический перенос их просто невозможен, а часто и недопустим.

Графа 5 также выглядит ненужной. Наименование завода-изготовителя выглядит как атавизм плановой системы. В современных условиях логичнее выглядело бы название графы "Поставщик" (это, скорее всего, будет не сам завод, а оптовый продавец оборудования), хотя для некоторых изделий мог бы быть указан предпочтительный изготовитель.

И последнее. Чрезвычайно нетехнологичной является и сама форма Спецификации. Разграфка на строки фиксированной высоты нужна только тем, кто пишет вручную. Ну и оставили бы ее для таких целей, но добавили бы заветное "допускается...", для автоматизированной подготовки. Различные формы основной надписи для первого и последующего листов Спецификации, взятые из общего стандарта на текстовые документы, очень неудобны для автоматизированного вывода спецификации на печать. Системы подготовки отчетов из баз данных просто не предполагают, что где-то еще не додумались до такой простой вещи, как единая форма всех листов. Конечно, выкрутиться можно, но с излишним усложнением программ.

Для чего мы занимались бесполезной критикой действующего нормативного документа, который нужно просто исполнять? Да для того, чтобы предупредить молодых разработчиков, что бесполезно автоматизировать составление Спецификаций, придерживаясь буквы стандарта. Это просто опасно, особенно если попытаться заносить в базу данных записи, подобные описанию вентилятора по строкам формы, т. к. при любой сортировке база может быть испорчена.

Не вздумайте использовать графу *Код оборудования для индексации баз данных* — она, по большей части, будет оставаться пустой.

Обзор методик работы со Спецификациями

Как же составляют спецификации оборудования при реальном проектировании? Не беремся отвечать за полноту обзора методик, но выделим несколько самых распространенных.

Ручное составление

Несмотря на повсеместное распространение компьютеров, множество проектировщиков пишут Спецификации вручную. Это не их вина, а беда. Написать Спецификацию авторучкой при отсутствии хороших программ часто быстрее, чем с использованием плохих программ. Это позор всей нашей ИТ-отрасли[1], а прежде всего тем,

[1] ИТ — информационные технологии.

кто командует проектированием и строительством. Разве трудно в масштабах страны продумать единую структуру базы данных оборудования, взять за основу (или поручить паре студентов-дипломников создать новую) одну из существующих программ, довести ее до блеска и бесплатно распространять среди всех проектных организаций? Бесплатно, но при условии, что все сформированные фрагменты специализированных БД будут передаваться в единый фонд для общего пользования. Увы, при современной политико-экономической ситуации это является утопией. Зачем "им" это надо?

Использование текстовых процессоров

Во многих проектных организациях "рисуют" Спецификации с помощью таких "подходящих" программ, как MS Word. Это уже прогресс, но такой же, как переход от каменного топора к бронзовому, только в XXI веке. Получаются документы, отпечатанные "красивым шрифтом", можно использовать заготовки, шаблоны, буфер обмена и пр. Самое ужасное, что эту технологию иногда внедряют специалисты, обязанные заниматься автоматизацией проектирования, сводящие свою роль к подготовке шаблонов бланков спецификаций. Это уже позор для таких "специалистов". К пользователям претензий быть не может, они просто не понимают, как "дурят нашего брата" (чаще сестру). Вред таких решений в том, что информация в текстовых документах не структурирована, ее нельзя извлечь для использования в базах данных, к которым неизбежно придется переходить. В некоторых проектных организациях такая технология поддерживается на основании мифов о том, что заказчикам могут понадобиться спецификации, сделанные в MS Word.

Рисование в AutoCAD

Выполняется также из-за того, что пользователям не предоставлено удобных средств. В системе AutoCAD удобно нарисовать и повторно использовать формы спецификаций или автоматически рисовать готовые спецификации, содержание которых наполнено в специальных программах — исключительно для формального соблюдения требований стандарта к форме Спецификации. Но написание текста спецификации вручную текстовыми примитивами системы AutoCAD — все тот же "пещерный век".

Использование электронных таблиц

Подготовка Спецификации в программах, подобных MS Excel, большой шаг вперед, сопоставимый с изобретением колеса. В Excel можно и вести примитивную базу оборудования, и готовить прекрасные итоговые документы, и автоматизировать определение некоторых вторичных "количеств" — например, пересчет объемов тепловой изоляции при изменении протяженности трубопроводов. При правильной организации структуры таблиц (не так, как в табл. 32.1, а так, как в табл. 32.2) банк данных оборудования из Excel можно экспортировать в настоящие базы данных. Изворотливые люди успешно сочетают Excel и систему AutoCAD. Но при всей прогрессивности такой технологии, она остается "домашней", т. е. рассчитанной на личное пользование или на работу в небольшой группе.

Использование настольных СУБД

Следующий шаг на пути прогресса — переход к работе в настольных системах управления базами данных, таких как MS Access. Хотя в этой системе может работать и обычный инженер, это уже переход к использованию настоящих баз данных. В Access можно успешно готовить качественные спецификации и вести полноценный банк данных. Недостаток этого банка данных в том, что он может использоваться только на локальном компьютере или на файловом сервере со всеми вытекающими проблемами. Наработанный в Access банк данных можно легко использовать в более совершенных системах.

Специализированные программы

Существует много специализированных программ, особым достоинством которых считается возможность автоматизированного составления спецификаций. Действительно, иногда это делается. Беда в том, что такие программы обычно занимаются автоматизацией проектирования отдельных "узких мест", да еще часто с очень специфичными задачами, да еще часто с использованием нестандартных способов хранения данных. Например, проектировщик наружных сетей водопровода и канализации может воспользоваться программой построения продольного профиля, которая составит и спецификацию на "опрофилированные" участки трассы. Но задачу составления спецификации на весь проект эта программа не решит — будет сэкономлено несколько минут на подсчет длины трассы.

Не отвергая возможностей таких программ в качестве дополнительного вспомогательного инструментария, мы продолжим поиски приемлемых решений.

Автоматическое определение объемов работ

В последнее время появились высококлассные продукты, при использовании которых якобы возможно автоматическое составление спецификаций. Впрочем, такая возможность декларировалась всегда — для этого в системе AutoCAD были предусмотрены блоки с атрибутами и возможность экспорта атрибутов в текстовые файлы. Эти файлы можно было впоследствии обработать, в том числе для составления спецификаций. Современные приложения на базе AutoCAD используют гораздо более развитые возможности, но до полной автоматизации[1] еще очень и очень далеко. Объемы работ (количество штук, квадратных и кубических метров) с помощью современных приложений определить можно, и это достаточно просто, но при непременном условии — все реальные объекты должны быть смоделированы или нарисованы с помощью этих приложений.

Например, мы легко могли бы дополнить программу рисования трубопроводов функцией, присоединяющей к изображению трубопровода расширенные данные со ссылкой на запись в базе данных с информацией, требуемой для заполнения граф спецификации (или просто присоединять эту информацию в виде расширенных данных), а затем извлекать обобщенные данные и выводить в требуемой форме.

---

[1] Пока действует знакомый принцип автоматизации — "нажал на кнопку и вся спина мокрая".

Но автоматический подсчет объемов и составление Спецификации — разные вещи. Спецификация — очень важный документ, по которому определяется стоимость строительства и комплектация оборудования. Текст каждой позиции должен иметь точную формулировку. Пытаясь облегчить работу по подсчету объемов работ, мы будем значительно усложнять рисование. Мы уже должны будем задавать не просто достаточный для рисования диаметр трубопровода, но и конкретную марку труб, откуда-то выбирая ее из обширного сортамента, да еще с учетом того, что для разных разделов проекта применяются разные формулировки заказа. И все усилия будут напрасными, т. к. при двухмерном черчении мы все равно не отобразим все участки трубопровода, или они могут оказаться отображенными в разных файлах.

При трехмерном моделировании ситуация кажется более оптимистичной — будут смоделированы все участки. Сложности с привязкой формулировок остаются, но они преодолимы. Для этого необходимо, чтобы программа позволяла удобно вести базу данных изделий и материалов, причем не так, как принято "где-то на Западе", а так, как требуется нам по действующим стандартам. Кроме того, мы должны моделировать свой объект на точной подоснове. Допустим, мы проектируем реконструкцию котельной. Кто даст проектировщику технологических трубопроводов трехмерную модель строительной части? Такое возможно только в эротическом сне!

Предположим (только предположим), что "свихнувшийся с ума" от избытка средств заказчик оплатит и эти работы. Всем, кто думает, что модель здания будет точно соответствовать действительности, рекомендуем зайти в любое промышленное здание и попытаться его обмерить, чтобы сделать полную модель, по которой можно построить точные модели сантехники, вентиляции и электрики. При условии, что за все ошибки в последующих определениях объемов работ придется расплачиваться из собственного кармана.

Даже если мы проектируем с нуля новое здание, далеко не всегда можно точно наметить трассы всех трубопроводов и воздуховодов. Довольно часто применяется нестандартизированное оборудование, которое на момент проектирования объекта еще даже не сконструировано или "установочные" чертежи на которое недоступны. Теоретически так проектировать нельзя, а практически приходится делать постоянно. В зависимости от сложности решений фундаменты и обвязка оборудования могут разрабатываться на дополнительных этапах, но часто выполняются "по месту и по соображению". Для этого необходимо, чтобы такие объемы работ, с некоторым запасом на незнание, были бы учтены в сметах и спецификациях.

Кроме того, в спецификациях должно быть учтено многое, не нарисованное или не отраженное в модели. Это и многочисленные подразумеваемые мелкие детали, и различные работы, например "разборка и восстановление покрытия", "резервный насос, хранящийся на складе", "врезка в действующие сети" и т. п. Некоторые записи в спецификации, например "испытание стыков физическими методами контроля", могут впоследствии спасти проектировщика от тюрьмы.

Вообще составление спецификаций, как и "рисование" любых денежных документов в строительстве, является не столько техническим, сколько творческим процессом. Полная его автоматизация, на наш взгляд, сравнима с полной автоматизацией машинного перевода, с такими же кажущимися успехами. Некоторые рутинные операции автоматизировать можно, а полностью — вряд ли.

Опытные проектировщики считают объемы очень быстро, у каждого есть своя методика. "Тетка" в должности руководителя группы всегда перепроверит вручную подготовленную "девочкой" Спецификацию и найдет массу упущений.

Мы не будем зацикливаться на бесплодных попытках автоматизированного заполнения графы "Количество", а сосредоточимся на максимальном облегчении подготовки текста документации.

Организация банка данных по оборудованию, изделиям и материалам

Итак, для составления Спецификаций необходим банк данных оборудования, изделий и материалов. Технические вопросы на современном уровне решаются достаточно просто, сложнее с организационными. Идея единого на всю страну банка данных осталась утопией. В одной проектной организации вести общий банк данных вполне реально. В прежние времена в проектных организациях существовали подразделения, занимавшиеся оборудованием (связи с заводами, коллекционирование "номенклатур", паспортов и чертежей), но они не имели почти никакого технического оснащения. Теперь все можно автоматизировать, но такие службы во многих организациях ликвидированы по экономическим причинам. Несомненно, эту глупость придется исправлять, но пока многие проектировщики вынуждены заниматься упоминавшейся ранее самодеятельностью. Играет роль и конкуренция внутри организации. Кто-то будет заниматься черновой работой по формированию банка, а кто-то будет "снимать пенки", используя чужой труд и хорошо при этом зарабатывая. Эта проблема решаемая, но при условии заинтересованности руководителей организации.

Программные решения должны быть масштабируемыми и позволять работать любым способом — от "частных банков" у каждого исполнителя до единого на организацию. Во всяком случае, структура данных и программные средства должны быть унифицированными, чтобы в лучшие времена можно было объединить мелкие базы данных.

Сразу следует забыть про милые детские глупости с Word и Excel. Более серьезные и популярные ранее решения с базами данных в формате DBF или Paradox также лучше забыть — это уже прошлый век. Вполне возможно использование баз данных в формате Access с последующей миграцией на Microsoft SQL Server. СУБД масштаба Oracle нам не понадобится — с банком данных в объеме максимум в несколько десятков тысяч записей[1] справится любая система. Файл-серверную систему применять также не стоит — только SQL-сервер.

Решения возможны разные, но в своей системе мы остановимся на клоне Firebird СУБД Interbase. Причины такого выбора:

- сервер Firebird (по крайней мере, до грядущего объединения с другим клоном Interbase — Yaffild) является бесплатным;
- сервер очень прост и надежен в эксплуатации и пользователь может не подозревать о его деятельности;
- Firebird можно установить на UNIX-сервере, с клиентами, работающими под Windows;

[1] Количество записей в сметных банках данных по всей мыслимой номенклатуре работ не превышает миллиона.

- имеется специальная версия Firebird Embedded Server, позволяющая, положив рядом с основным приложением несколько DLL, работать с базами данных на локальной машине как через настоящий сервер, не изменив ни строчки программного кода.

Использование механизма SQL-запросов и приличного набора инструментальных средств для администрирования таких баз данных позволит импортировать данные из других источников.

Структура базы данных

Проектирование структуры базы данных — важнейший этап. В нашем случае важно не увлечься излишней *нормализацией*, столь любимой разработчиками баз данных.

Создание базы данных Interbase мы выполняем с помощью великолепной программы IBExpert. Не останавливаясь на деталях работы с этим продуктом, приведем только SQL-запрос для создания базы данных (листинг 32.1). Несущественные детали мы опускаем ради экономии места.

Листинг 32.1. SQL-запрос для создания базы данных оборудования

```
SET NAMES WIN1251;
CREATE DATABASE 'RUSO.GDB'
USER 'SYSDBA' PASSWORD 'masterkey'
PAGE_SIZE 1024
DEFAULT CHARACTER SET WIN1251;
/**** Генераторы ******************************************************/
CREATE GENERATOR GEN_DIC_PLANTS_ID;
SET GENERATOR GEN_DIC_PLANTS_ID TO 200;
CREATE GENERATOR GEN_DIC_UNITS_ID;
SET GENERATOR GEN_DIC_UNITS_ID TO 11;
CREATE GENERATOR GEN_OBOR_ID;
SET GENERATOR GEN_OBOR_ID TO 1228;
CREATE GENERATOR GEN_SO_FRAGMENTS_ID;
SET GENERATOR GEN_SO_FRAGMENTS_ID TO 20;
/****                      Таблицы                                  ****/
/****          Справочник заводов-изготовителей                     ****/
CREATE TABLE DIC_PLANTS (
    ID_P         INTEGER NOT NULL,
    ID_PARENT_P  INTEGER,
    NAME_P       VARCHAR(250) CHARACTER SET WIN1251,
    ADRESS       VARCHAR(120) CHARACTER SET WIN1251,
    URL          VARCHAR(200) CHARACTER SET WIN1251
);
/****          Справочник единиц измерения                          ****/
CREATE TABLE DIC_UNITS (
    ID_U       INTEGER NOT NULL,
    NAME_UNIT  CHAR(10) CHARACTER SET WIN1251 NOT NULL
);
```

```
/****          Таблица типовых фрагментов                          ****/
CREATE TABLE SO_FRAGMENTS (
    ID_F          INTEGER NOT NULL,
    ID_PARENT_F   INTEGER NOT NULL,
    NAME_F        VARCHAR(500) CHARACTER SET WIN1251,
    POS_F         VARCHAR(10) CHARACTER SET WIN1251,
    CODE_IN_OBOR  INTEGER NOT NULL
);
/****          Главная таблица с оборудованием                     ****/
CREATE TABLE SO_OBOR (
    ID          INTEGER NOT NULL,
    ID_PARENT   INTEGER NOT NULL,
    NAME_OBOR   VARCHAR(500) CHARACTER SET WIN1251,
    TIP_OBOR    VARCHAR(250) CHARACTER SET WIN1251,
    CODE_OBOR   VARCHAR(20) CHARACTER SET WIN1251,
    ZAVOD       VARCHAR(100) CHARACTER SET WIN1251,
    ED_IZM      VARCHAR(10) CHARACTER SET WIN1251,
    KOL_ED      NUMERIC(15,3),
    MASSA_ED    NUMERIC(15,4),
    PRIM        VARCHAR(10) CHARACTER SET WIN1251,
    PHOTO_LINK  VARCHAR(250) CHARACTER SET WIN1251,
    POS         VARCHAR(10) CHARACTER SET WIN1251
);
/****                    Просмотры                                 ****/
CREATE VIEW FRAGMENT(
    NAME_OBOR,
    F_TIP_OBOR,
    F_CODE_OBOR,
    F_ZAVOD,
    F_ED_IZM,
    F_KOL_ED,
    F_MASSA_ED,
    F_PRIM,
    F_PHOTO_LINK)
AS
select  POS_F ||' ' || NAME_OBOR, TIP_OBOR, CODE_OBOR, ZAVOD, ED_IZM, KOL_ED,
MASSA_ED, PRIM, PHOTO_LINK
 from so_obor, so_fragments
where (so_fragments.code_in_obor = so_obor.id);
/****                     Первичные ключи                          ****/
ALTER TABLE DIC_PLANTS ADD CONSTRAINT PK_DIC_PLANTS PRIMARY KEY (ID_P);
ALTER TABLE DIC_UNITS ADD CONSTRAINT PK_DIC_UNITS PRIMARY KEY (ID_U);
ALTER TABLE SO_FRAGMENTS ADD CONSTRAINT PK_SO_FRAGMENTS PRIMARY KEY (ID_F);
ALTER TABLE SO_OBOR ADD CONSTRAINT PK_SO_OBOR PRIMARY KEY (ID);
/****                      Триггеры                                ****/
SET TERM ^;
CREATE TRIGGER DIC_PLANTS_BI FOR DIC_PLANTS
ACTIVE BEFORE INSERT POSITION 0
AS
BEGIN
```

```
  IF (NEW.ID_P IS NULL) THEN
    NEW.ID_P = GEN_ID(GEN_DIC_PLANTS_ID,1);
END
^
/* Trigger: DIC_UNITS_BI */
CREATE TRIGGER DIC_UNITS_BI FOR DIC_UNITS
ACTIVE BEFORE INSERT POSITION 0
AS
BEGIN
  IF (NEW.ID_U IS NULL) THEN
    NEW.ID_U = GEN_ID(GEN_DIC_UNITS_ID,1);
END
^
/* Trigger: INS_OBOR_TRIGGER */
CREATE TRIGGER INS_OBOR_TRIGGER FOR SO_OBOR
ACTIVE BEFORE INSERT POSITION 0
AS
begin
  new.id = gen_id(gen_obor_id, 1);
end
^

/* Trigger: SO_FRAGMENTS_BI */
CREATE TRIGGER SO_FRAGMENTS_BI FOR SO_FRAGMENTS
ACTIVE BEFORE INSERT POSITION 0
AS
BEGIN
  IF (NEW.ID_F IS NULL) THEN
    NEW.ID_F = GEN_ID(GEN_SO_FRAGMENTS_ID,1);
END
^

SET TERM; ^


/* Пояснения к полям */
DESCRIBE FIELD ADRESS TABLE DIC_PLANTS 'Адрес завода';
DESCRIBE FIELD ID_P TABLE DIC_PLANTS 'Код';
DESCRIBE FIELD ID_PARENT_P TABLE DIC_PLANTS 'Код предка';
DESCRIBE FIELD NAME_P TABLE DIC_PLANTS 'Наименование';
DESCRIBE FIELD URL TABLE DIC_PLANTS 'Ссылка на сайт с номенклатурой';
DESCRIBE FIELD ID_U TABLE DIC_UNITS 'Код';
DESCRIBE FIELD NAME_UNIT TABLE DIC_UNITS 'Наименование';
DESCRIBE FIELD CODE_IN_OBOR TABLE SO_FRAGMENTS 'Код записи в таблице оборудования';
DESCRIBE FIELD ID_F TABLE SO_FRAGMENTS'Код записи';
DESCRIBE FIELD ID_PARENT_F TABLE SO_FRAGMENTS 'Код предка';
DESCRIBE FIELD NAME_F TABLE SO_FRAGMENTS 'Наименование';
DESCRIBE FIELD POS_F TABLE SO_FRAGMENTS 'Позиция в фрагменте';
DESCRIBE FIELD CODE_OBOR TABLE SO_OBOR 'Код оборудования';
DESCRIBE FIELD ED_IZM TABLE SO_OBOR 'Изм';
DESCRIBE FIELD ID TABLE SO_OBOR 'Код';
```

```
DESCRIBE FIELD ID_PARENT TABLE SO_OBOR 'Код владельца';
DESCRIBE FIELD KOL_ED TABLE SO_OBOR 'Количество';
DESCRIBE FIELD MASSA_ED TABLE SO_OBOR 'Масса ед.';
DESCRIBE FIELD NAME_OBOR TABLE SO_OBOR 'Наименование оборудования';
DESCRIBE FIELD PHOTO_LINK TABLE SO_OBOR 'Ссылка на файл с изображением';
DESCRIBE FIELD POS TABLE SO_OBOR 'Позиция';
DESCRIBE FIELD PRIM TABLE SO_OBOR 'Примечание';
DESCRIBE FIELD TIP_OBOR TABLE SO_OBOR 'Тип, марка';
DESCRIBE FIELD ZAVOD TABLE SO_OBOR 'Завод';
```

Напомним читателям, что мы уже сравнивали SQL-запросы в мире СУБД с выражениями AutoLISP в командной строке AutoCAD. Выполнив этот SQL-запрос, мы создадим базу данных Interbase с несколькими таблицами.

Знатоки SQL наверняка сообразили, что на самом деле мы просто экспортировали из готовой базы данных ее структуру. Кроме того, видно, что вопреки теории баз данных у нас имеется дублированная информация — заводы-изготовители и единицы измерения заносятся в виде текстов, а не связок по кодам, и таблицы заводов и единиц измерения используются просто в виде справочников, облегчающих ввод повторяющихся данных. Для справочника заводов мы предусмотрели поля адреса и URL[1], не требуемые в Спецификации, но облегчающие работу проектировщиков. По URL, например, можно прямо из программы перейти на сайт завода для уточнения номенклатуры.

В основной таблице имеется поле ссылки на файл с изображением оборудования. Разве плохо, если "девочка" сможет по фотографии узнать, чем вентиль отличается от задвижки?

Разработка программы

К сожалению, у нас нет возможности рассказать, как делается такая программа — наберется несколько глав, и все "не про AutoCAD". Как разрабатывать приложения, работающие с базами данных, написано во множестве книг. Мы остановимся на том, как должна работать такая программа.

Программу подготовки спецификаций **ruSO** мы делаем в виде самостоятельного приложения. Она может работать и "сама по себе", но может быть запущена и из меню системы AutoCAD. При самостоятельной работе в ruSO пользователь или пополняет базу данных оборудования, или формирует рабочую спецификацию, или делает это одновременно. После завершения формирования рабочей спецификации она может быть отпечатана прямо из программы в виде отчета, или может быть "нарисована" в AutoCAD, если "рисованная" спецификация больше нравится.

В принципе, для такой программы даже не важно, как она сделана и с каким банком данных работает — можно даже вообще не иметь базы данных оборудования, а просто редактировать таблицы рабочих спецификаций, используя прежние разработки и прототипы. Взаимодействия с системой AutoCAD практически нет никако-

[1] URL — Uniform Resource Locator, унифицированный указатель информационного ресурса, т. е. стандартизованная строка символов, указывающая местонахождение документа в сети Интернет.

го, хотя при заполнении графы **Количество** возможно использование результатов полуавтоматического определения объемов работ.

Имеется вариант программы, специально сделанный для "единоличников" — проектировщиков, предпочитающих вообще не связываться с коллегами, а все свое держать на собственном компьютере. Эта версия программы (рис. 32.1) вообще не использует единой базы данных, а все спецификации (рабочие, справочники, фрагменты и шаблоны) хранятся в одинаковых CSV-файлах (листинг 32.2). Файлы находятся в специальной ветви рабочего каталога, навигация по которой осуществляется так же, как и по каталогам классификатора слоев.

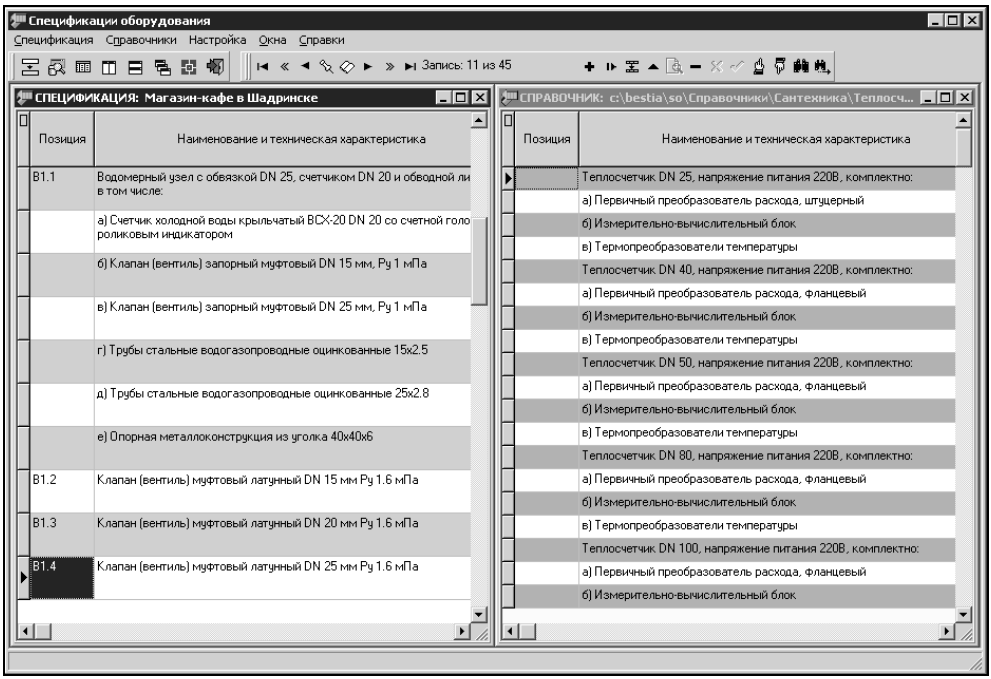


Рис. 32.1. Упрощенная версия программы подготовки спецификаций оборудования

Пользователь открывает два окна, в одном из которых будет формировать рабочую спецификацию, а другое будет использовать как справочник оборудования. Рабочую спецификацию или ее фрагменты также можно сохранять в качестве справочников, фрагментов или шаблонов.

Преимущества и недостатки такого подхода примерно такие же, как и при формировании документов в текстовом редакторе с использованием множества файлов — опасность запутаться с большим количеством файлов при простоте единообразных приемов работы. Имеется определенное удобство в возможности синхронизации с другими компьютерами по датам обновления файлов — отыскать изменения в единой базе данных без специальных полей невозможно. У пользователя существует две возможности — иметь один или несколько больших справочников и долго выбирать в них нужные позиции или иметь много маленьких справочников в дереве каталогов

и постоянно рыться в этом дереве. И то, и другое сначала кажется удобным, но по мере роста квалификации пользователь начинает понимать, что все это плохо и переходит на работу с базой данных, разделяя труд по ее сопровождению с коллегами.

Данные, накопленные в отдельных файлах, легко можно импортировать в настоящую базу данных и перейти на работу с "настоящей" программой.

Все варианты программы мы будем разрабатывать в среде Borland Delphi. Эта система позволяет очень комфортно разрабатывать приложения любой сложности для работы с базами данных. Помимо большого набора стандартных компонентов можно использовать множество библиотек независимых разработчиков. Не вдаваясь в подробности, перечислим только основные компоненты, которые мы будем использовать (некоторые мы уже применяли при разработке редактора таблиц):

- библиотеку EhLib Дмитрия Большакова;
- компонент `kbmMemTable` для работы с CSV-файлами;
- компоненты из библиотеки VirtualTreeView для отображения базы данных в виде дерева;
- библиотеку FibPlus для работы с базой данных Firebird (эта библиотека коммерческая, но она настолько упрощает работу, что нам выгоднее приобрести ее, чем возиться с бесплатными компонентами).

Формирование спецификации для конкретного объекта

Спецификацию для конкретного объекта (будем далее называть ее *рабочей спецификацией*) целесообразно сохранять в виде локальных таблиц в текстовом формате CSV. Для того чтобы отличать наши файлы спецификаций оборудования от других таблиц, мы будем присваивать им расширение ruso. В *главе 16* мы уже разработали функции для заполнения в системе AutoCAD таблиц данными из файлов такого формата. Визуальное редактирование подобных файлов очень удобно с использованием компонента `kbmMemTable`. Класс `TkbmMemTable` является "потомком" стандартного класса `TdataSet` и полностью совместим со всеми визуальными компонентами для редактирования данных. Во время работы данные полностью находятся в памяти и это очень важно, т. к. позволяет легко управлять порядком расположения строк — при необходимости любую строку можно перенести в любое место. При желании можно также отредактировать файл рабочей спецификации в обычном текстовом редакторе.

Замечание

Не следует формировать рабочую спецификацию из ссылок на записи в базе данных оборудования — это первым делом приходит в голову специалистам по СУБД.

Если, например, в момент формирования спецификации изделие в банке данных именовалось "вентиль", а потом его стали именовать "клапан", то в рабочей спецификации и должен навсегда (или до внесения изменений) остаться именно "вентиль", т. к. рабочая спецификация является юридическим[1] и финансовым документом. Записи должны имен-

[1] Не верите? Со спецификациями оборудования (в числе прочих документов) разбирается прокуратура, расследуя причины аварий объектов. В этом случае очень важно, что было записано в спецификации во время проектирования, а не то, что могло бы быть записано сейчас.

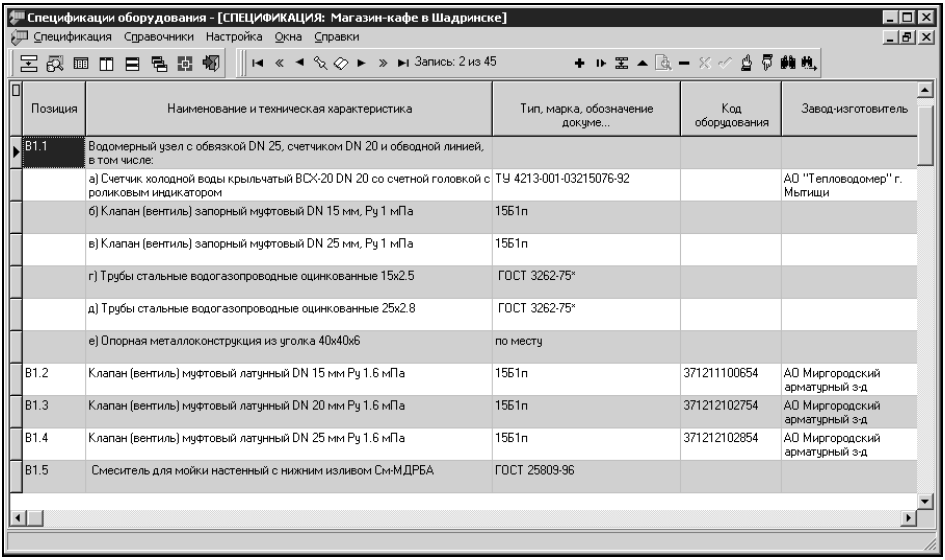
но копироваться и полностью терять связь с "родным гнездом". В рабочую спецификацию могут заноситься и отсутствующие в банке оборудования записи.

Структура таблицы полностью соответствует структуре готовой Спецификации. В первой строке файла записываются имена полей, во всех последующих — данные. Ограничений на имена полей нет. Поля разделяются запятыми, строковые данные записываются в двойных кавычках. Фрагмент файла рабочей спецификации приведен в листинге 32.2 (перенос строк выполнен по формату книги, для выделения строк файла в листинг вставлены пустые строки).

Листинг 32.2. Фрагмент файла рабочей спецификации

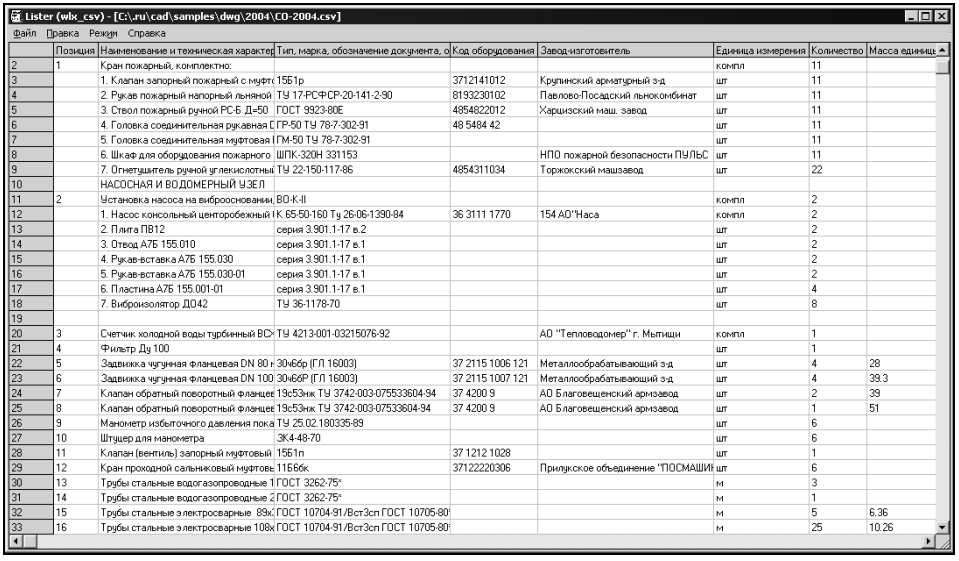
```
"Позиция","Наименование и техническая характеристика","Тип, марка, обозначение
документа, опросного листа","Код оборудования","Завод-изготовитель","Единица
измерения","Количество","Масса единицы, кг","Примечание",

">","%%%%uВОДОПРОВОД В1","","","","",,,"",

"В1.1","Водомерный узел с обвязкой DN 80, счетчиком DN 50 и обводной линией, в том
числе:","","","","компл","1",,,

"","а) Счетчик холодной воды крыльчатый ВСХ-50 DN 50 со счетной головкой с
роликовым индикатором","ТУ 4213-001-03215076-92","","АО ""Тепловодомер"" г.
Мытищи","шт","1",,,

"","б) Задвижка клиновая фланцевая DN 80 мм, Ру 1 мПа","31ч6бр","37 2113
1029","Георгиевский арматурный з-д ","шт","3",,,

"","в) Клапан (вентиль) запорный муфтовый DN 25 мм, Ру 1 мПа","15Б1п","37 1212
1028","","шт","1",,,

"","г) Кран проходной сальниковый муфтовый DN 15 Ру 1.0
Мпа","11Б6бк","37122220306","Прилукское объединение ""ПОСМАШИНА""","шт","1",,,

"","д) Трубы стальные водогазопроводные 15х2.8"," ГОСТ 3262-75*","","","м","0.5",,,

"","е) Трубы стальные водогазопроводные 25х3.2"," ГОСТ 3262-75*","","","м","0.5",,,

"","ж) Трубы стальные водогазопроводные 50х3.5"," ГОСТ 3262-75*","","","м","1",,,

"","з) Трубы стальные электросварные 89х3.0 ГОСТ 10704-91*/В ст3сп
ГОСТ 10705-80*","","","","","м","10",,,

"","и) Манометр МПУ-3-16","ТУ 25.02.180335-89","","Манометровый з-д, г.
Томск","шт","1",,,

"","к) Штуцер для манометра","ЗК4-48-70","","","шт","1",,,

"","л) Опорная металлоконструкция из уголка 63х63х6","по месту","","","кг","30",,,
```

Примерный вид окна программы во время редактирования рабочей спецификации показан на рис. 32.2.



Позиция	Наименование и техническая характеристика	Тип, марка, обозначение докуме...	Код оборудования	Завод-изготовитель
В1.1	Водомерный узел с обвязкой DN 25, счетчиком DN 20 и обводной линией, в том числе:			
	а) Счетчик холодной воды крыльчатый ВСХ-20 DN 20 со счетной головкой с роликовым индикатором	ТУ 4213-001-03215076-92		АО "Тепловодомер" г. Мытищи
	б) Клапан (вентиль) запорный муфтовый DN 15 мм, Ру 1 мПа	15Б1п		
	в) Клапан (вентиль) запорный муфтовый DN 25 мм, Ру 1 мПа	15Б1п		
	г) Трубы стальные водогазопроводные оцинкованные 15х2.5	ГОСТ 3262-75*		
	д) Трубы стальные водогазопроводные оцинкованные 25х2.8	ГОСТ 3262-75*		
	е) Опорная металлоконструкция из уголка 40х40х6	по месту		
В1.2	Клапан (вентиль) муфтовый латунный DN 15 мм Ру 1.6 мПа	15Б1п	371211100654	АО Миргородский арматурный з-д
В1.3	Клапан (вентиль) муфтовый латунный DN 20 мм Ру 1.6 мПа	15Б1п	371212102754	АО Миргородский арматурный з-д
В1.4	Клапан (вентиль) муфтовый латунный DN 25 мм Ру 1.6 мПа	15Б1п	371212102854	АО Миргородский арматурный з-д
В1.5	Смеситель для мойки настенный с нижним изливом См-МДРБА	ГОСТ 25809-96		

Рис. 32.2. Редактирование рабочей спецификации



	Позиция	Наименование и техническая характер	Тип, марка, обозначение документа, о	Код оборудования	Завод-изготовитель	Единица измерения	Количество	Масса единиц
2	1	Кран пожарный, комплектно:				компл	11	
3		1. Клапан запорный пожарный с муфт	15Б1р	3712141012	Крупинский арматурный з-д	шт	11	
4		2. Рукав пожарный напорный льняной	ТУ 17-РСФСР-20-141-2-90	8193230102	Павлово-Посадский льнокомбинат	шт	11	
5		3. Ствол пожарный ручной РС-Б Д=50	ГОСТ 9923-80Е	4854822012	Харцизский маш. завод	шт	11	
6		4. Головка соединительная рукавная	ГР-50 ТУ 78-7-302-91	48 5484 42		шт	11	
7		5. Головка соединительная муфтовая	ГМ-50 ТУ 78-7-302-91			шт	11	
8		6. Шкаф для оборудования пожарного	ШПК-320Н 331153		НПО пожарной безопасности ПУЛЬС	шт	11	
9		7. Огнетушитель ручной углекислотны	ТУ 22-150-117-86	4854311034	Торжокский машзавод	шт	22	
10		НАСОСНАЯ И ВОДОМЕРНЫЙ УЗЕЛ						
11	2	Установка насоса на виброосновании	ВО-К-II			компл	2	
12		1. Насос консольный центробежный	К 65-50-160 Ту 26-06-1390-84	36 3111 1770	154 АО"Наса	компл	2	
13		2. Плита ПВ12	серия 3.901.1-17 в.2			шт	2	
14		3. Отвод А7Б 155.010	серия 3.901.1-17 в.1			шт	2	
15		4. Рукав-вставка А7Б 155.030	серия 3.901.1-17 в.1			шт	2	
16		5. Рукав-вставка А7Б 155.030-01	серия 3.901.1-17 в.1			шт	2	
17		6. Пластина А7Б 155.001-01	серия 3.901.1-17 в.1			шт	4	
18		7. Виброизолятор ДО42	ТУ 36-1178-70			шт	8	
19								
20	3	Счетчик холодной воды турбинный ВСХ	ТУ 4213-001-03215076-92		АО "Тепловодомер" г. Мытищи	компл	1	
21	4	Фильтр Ду 100				шт	1	
22	5	Задвижка чугунная фланцевая DN 80 к	30ч6бр (ГЛ 16003)	37 2115 1006 121	Металлообрабатывающий з-д	шт	4	28
23	6	Задвижка чугунная фланцевая DN 100	30ч6бР (ГЛ 16003)	37 2115 1007 121	Металлообрабатывающий з-д	шт	4	39.3
24	7	Клапан обратный поворотный фланцев	19с53нж ТУ 3742-003-07553604-94	37 4200 9	АО Благовещенский армзавод	шт	2	39
25	8	Клапан обратный поворотный фланцев	19с53нж ТУ 3742-003-07533604-94	37 4200 9	АО Благовещенский армзавод	шт	1	51
26	9	Манометр избыточного давления пока	ТУ 25.02.180335-89			шт	6	
27	10	Штуцер для манометра	ЗК4-48-70			шт	6	
28	11	Клапан (вентиль) запорный муфтовый	15Б1п	37 1212 1028		шт	1	
29	12	Кран проходной сальниковый муфтовы	11Б6бк	37122220306	Прилукское объединение "ПОСМАШИ	шт	6	
30	13	Трубы стальные водогазопроводные	ГОСТ 3262-75*			м	3	
31	14	Трубы стальные водогазопроводные 2	ГОСТ 3262-75*			м	1	
32	15	Трубы стальные электросварные 89х	ГОСТ 10704-91/Вст3сп ГОСТ 10705-80			м	5	6.36
33	16	Трубы стальные электросварные 108х	ГОСТ 10704-91/Вст3сп ГОСТ 10705-80			м	25	10.26

Рис. 32.3. Просмотр рабочей спецификации в Total Commander

Очень удобно, что спецификацию в формате CSV можно просматривать с помощью очень удобного "Lister-plugin[1]" wlx_csv.wlx[2] для файлового менеджера Total

[1] plugin, plug-in — дополнительно подключаемый программный блок.

[2] atlanoff.narod.ru.

Commander (рис. 32.3) простым нажатием клавиши <F3>. Так как у наших спецификаций будет нестандартное расширение ruso, в файле настроек wlx_csv.ini необходимо исправить строку `ext=.csv;.tab;.ruso`.

Формирование рабочей спецификации

В самом примитивном случае рабочую спецификацию можно просто написать вручную, используя форму как электронный бланк. Таким образом поступают, пока базы данных оборудования вообще еще нет.

В рабочую спецификацию можно вставить строки из другой спецификации или использовать готовую спецификацию в качестве прототипа.

По мере формирования банка данных оборудования появляется возможность экспорта в рабочую спецификацию записей из основной базы данных оборудования — отдельными записями, группами выделенных записей или целыми фрагментами. База данных оборудования отображается в виде структурированного дерева (рис. 32.4) или в виде таблицы. При необходимости базу оборудования можно на лету дополнять и изменять структуру дерева перетаскиванием узлов. В базе могут быть не только записи оборудования, изделий и материалов, но и типовые заголовки разделов.

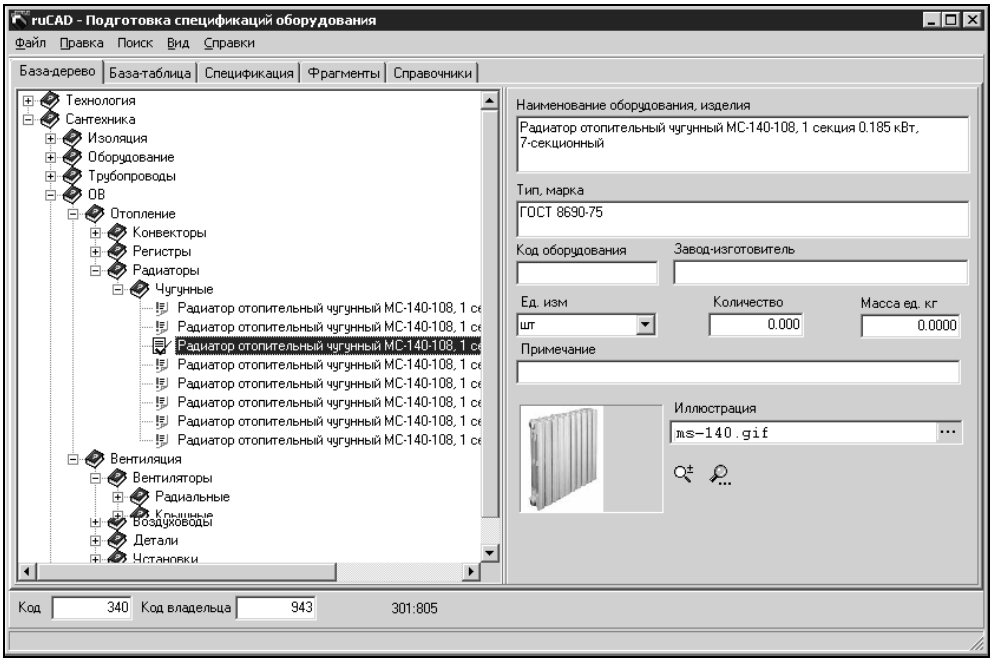


Рис. 32.4. Работа с базой данных оборудования

Значительно ускоряют работу типовые фрагменты — группы записей, физически находящихся в разных местах базы данных оборудования, но вносимых в рабочую

спецификацию за один прием. На рис. 32.2 видны записи, вставленные в виде типового фрагмента (позиция В1.1, водомерный узел). В некоторые типовые фрагменты могут включаться десятки записей. В виде фрагмента может быть сгруппировано, например, оборудование для типовой секции жилого дома.

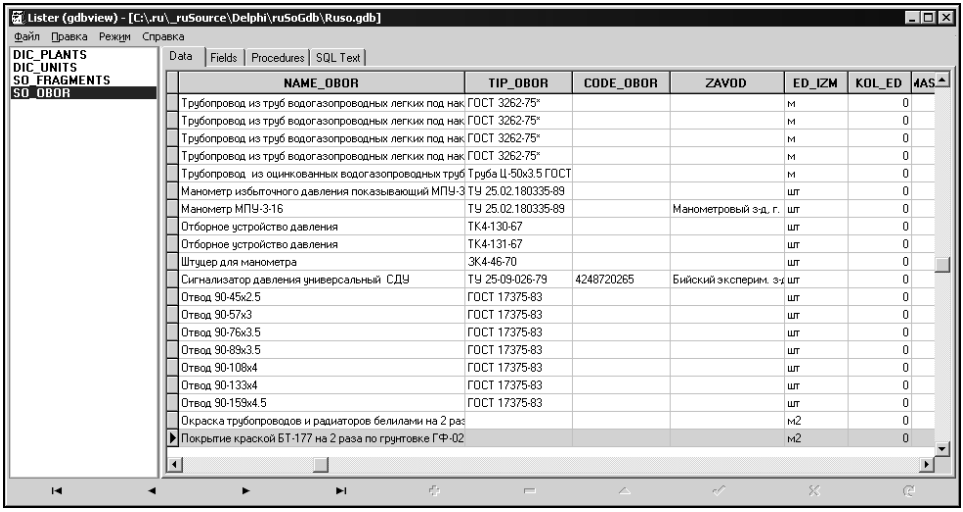
Оператор SQL из листинга 32.1

```
select  POS_F ||' ' || NAME_OBOR, TIP_OBOR, CODE_OBOR, ZAVOD, ED_IZM, KOL_ED,
MASSA_ED, PRIM, PHOTO_LINK
```

позволяет автоматически добавлять в графе наименования "подпозиции" оборудования внутри фрагмента. Отображенные в рис. 32.2 "номера" а), б) и т. п. в базе данных отсутствуют, но подставляются автоматически при формировании фрагмента с помощью языка SQL.

Замечание

Кстати, быстро просмотреть базу данных в формате Interbase теперь можно с помощью Lister-plugin GDBView.wlx[1] для Total Commander (рис. 32.5). Это возможно, даже если на компьютере вообще не установлен сервер Interbase — достаточно поместить в каталог Total Commander файлы GDS32.DLL и FIREBIRD.MSG из комплекта Firebird Embedded Server. Это намек тем, кто по незнанию боится использовать серверы баз данных.



NAME_OBOR	TIP_OBOR	CODE_OBOR	ZAVOD	ED_IZM	KOL_ED
Трубопровод из труб водогазопроводных легких под нак	ГОСТ 3262-75*			м	0
Трубопровод из труб водогазопроводных легких под нак	ГОСТ 3262-75*			м	0
Трубопровод из труб водогазопроводных легких под нак	ГОСТ 3262-75*			м	0
Трубопровод из труб водогазопроводных легких под нак	ГОСТ 3262-75*			м	0
Трубопровод из оцинкованных водогазопроводных труб	Труба Ц-50х3.5 ГОСТ			м	0
Манометр избыточного давления показывающий МПУ-3	ТУ 25.02.180335-89			шт	0
Манометр МПУ-3-16	ТУ 25.02.180335-89		Манометровый з-д, г.	шт	0
Отборное устройство давления	ТК4-130-67			шт	0
Отборное устройство давления	ТК4-131-67			шт	0
Штуцер для манометра	ЗК4-46-70			шт	0
Сигнализатор давления универсальный СДУ	ТУ 25-09-026-79	4248720265	Бийский эксперим. з-д	шт	0
Отвод 90-45х2.5	ГОСТ 17375-83			шт	0
Отвод 90-57х3	ГОСТ 17375-83			шт	0
Отвод 90-76х3.5	ГОСТ 17375-83			шт	0
Отвод 90-89х3.5	ГОСТ 17375-83			шт	0
Отвод 90-108х4	ГОСТ 17375-83			шт	0
Отвод 90-133х4	ГОСТ 17375-83			шт	0
Отвод 90-159х4.5	ГОСТ 17375-83			шт	0
Окраска трубопроводов и радиаторов белилами на 2 раз				м2	0
Покрытие краской БТ-177 на 2 раза по грунтовке ГФ-02				м2	0

Рис. 32.5. Просмотр базы данных оборудования в Total Commander

Таким образом, пользователь может очень быстро выполнить самую объемную и рутинную работу — формирование номенклатуры записей спецификации. Остается определить объемы работ и указать их в спецификации. О проблемах с автоматическим определением объемов мы уже достаточно подробно писали. Однако если действительно разработана полная автоматизация какой-то задачи, а это вполне воз-

[1] Все модули расширения Total Commander можно найти на прекрасном сайте **www.wincommander.ru**.

можно, то следует предусматривать составление спецификации в рамках этой задачи с выводом данных не в готовом виде, а в формат CSV. В этом случае полученные спецификации можно использовать в качестве фрагментов.

Пополнение и обслуживание банка данных

Редактирование и пополнение банка данных может выполняться одновременно с формированием рабочей спецификации. Имеется возможность импортирования данных из других форматов с помощью специальной утилиты. При размещении базы данных на удаленном сервере возможно одновременное использование и редактирование банка данных многими пользователями.

Проектировщики, недостаточно знакомые с технологиями баз данных, часто опасаются "хранения всех яиц в одной корзине" — т. е. боятся разом потерять все данные. Опасения эти, при использовании современных клиент-серверных технологий, излишни. База данных может легко автоматически резервироваться с заданной периодичностью. Конечно, совсем на произвол судьбы ее бросать нельзя. В проектной организации должен быть квалифицированный администратор баз данных — работа ему найдется не только с базой оборудования, но и с электронным архивом, который мы рассмотрим в *главе 33*.

Очень хорошо, если базу данных оборудования будет пополнять и следить за актуальностью данных специалист по оборудованию.

Вывод спецификации на бумагу

Конечным результатом должна быть спецификация оборудования, отпечатанная на бумаге. Вывод спецификации на бумагу возможен непосредственно из программы редактирования в виде так называемого *отчета*. Формы отчетов разрабатываются с помощью различных *генераторов отчетов*. Источниками данных для отчетов могут быть любые компоненты — наследники `TDataSet`. В своей системе мы используем очень удобный генератор отчетов FastReport.

Усложняет конструирование отчета нетехнологичная форма спецификации оборудования с различной основной надписью для первого и последующего листов. Мы считаем возможным пойти на отступление от стандарта и выводить одинаковую основную надпись на каждом листе спецификации, указывая в графе наименования чертежа "Спецификация оборудования (начало)", "Спецификация оборудования (продолжение)" и "Спецификация оборудования (окончание)" так же, как это предусмотрено стандартом для общих данных по рабочим чертежам. Кроме того, не выводятся внешние границы формата, т. к. их рисование при печати на форматированную бумагу не имеет никакого смысла.

Вывод отчета производится с предварительным просмотром и с возможностью настройки устройства для печати.

При необходимости на печать можно вывести черновой вариант спецификации прямо из сетки для отображения данных — для этого вообще не требуется программирование.

Для заполнения граф основной надписи программа подготовки спецификаций должна включать специальное диалоговое окно, представляющее урезанную версию Мастера, рассмотренного нами в *главе 21*. Данные для заполнения "штампа" сохраняются так же, как и у основного Мастера вычерчивания формата, что позволяет единообразно обрабатывать их в системе AutoCAD.

Рисование спецификации в AutoCAD

Хотя вывод спецификации в виде отчета очень удобен, многие пользователи предпочитают нарисовать ее в системе AutoCAD, а потом мучиться с постраничной печатью. Преимущество этого метода только в том, что рисуется *точная* форма спецификации, автоматически создается требуемое количество листов, в которые вписывается текст рабочей спецификации и заполняются графы основной надписи.

Рисование спецификации лучше выполнять в пространстве модели в отдельном файле. Масштаб чертежа лучше установить 1:1, а единицы рисунка — миллиметры, хотя все будет нарисовано пропорционально и при любых иных параметрах. Нарисованную спецификацию можно вывести на бумагу стандартными средствами AutoCAD, поочередно выводя на плоттер указанные рамкой листы.

От автоматического создания компоновок для каждого листа спецификации мы отказались, т. к. этот вариант пользователям не очень-то нравится, но значительно усложняет работу.

Программу для рисования спецификаций мы рассмотрим далее.

Подготовка форм

Для рисования спецификации нужны две формы — для первого и последующего листов. Формы спецификации оборудования мы нарисуем своей же программой рисования формата. "Местом жительства" для форм мы определим каталог %LocalData%\So\, в котором будут находиться файлы ru-so-first.dwg и ru-so-next.dwg.

В этом же каталоге, при необходимости, можно размещать данные для упрощенной версии программы.

Файлы форм включают атрибуты основной надписи — точно такие же, как и для программы рисования формата.

Общая схема работы

Целесообразно подготовить две функции — для запуска одной из программ подготовки спецификаций и для рисования спецификации. Запуск программы можно выполнять в модальном режиме (дожидаясь завершения работы) или в параллельном процессе. На уровне меню мы традиционно будем загружать и выполнять короткие программы. Запуск упрощенной версии редактора спецификаций показан в листинге 32.3.

Листинг 32.3. Файл ru_so_edit_lt.lsp

```
(defun START ()
  (ru-app-begin)
```

```
  (ru-so-run-app "ruSoCsv" t)
  (ru-app-end)
  (princ)
);_ end of defun
(START)
```

Функция для запуска редактора спецификаций приведена в листинге 32.4.

Листинг 32.4. Функция ru-so-run-app

```
(defun ru-so-run-app (exe_name is_modal / reason so_file so_key)
  (setq so_file  (ru-file-dwgname-type ".ruso")
        so_key   (ru-file-set-ext exe_name "")
        exe_name
                 (ru-file-bin exe_name)
 );_ end of setq
  (if (findfile exe_name)
    (progn
;; Записываем задание для программы
      (ru-reg-write-tmp-string so_key "Result" so_file)
;; Запускаем редактор спецификаций
      (ru-app-run exe_name is_modal)
      (if is_modal
        (progn
;; Выясняем причину выхода из программы
          (setq so_file (ru-reg-read-tmp-string so_key "Result" so_file)
                reason  (ru-reg-read-tmp-string so_key "ResultDial" "1")
         );_ end of setq
          (if (= reason "1")
;; Если был выполнен выход с рисованием спецификации,
;; то рисуем спецификацию
            (ru-so-draw-in-dwg so_file)
         );_ end of if
       );_ end of progn
     );_ end of if
   );_ end of progn
    (ru-msg-alert (strcat "Не найден \n" exe_name))
 );_ end of if
  (princ)
)
```

Программа для рисования выбранной спецификации оборудования без запуска редактора приведена в листинге 32.5.

Листинг 32.5. Файл ru_so_select_and_draw.lsp

```
(defun START (/ so_file)
 (ru-app-begin)
  (if (setq so_file (_ru-dlg-file
```

```
                    "Выбор файла спецификации оборудования"
                    (ru-user-work-dir) (getvar "DWGPREFIX")
                    "*.ruso" "" nil T nil
                  );_ end of _ru-dlg-file
    );_ end of setq
    (ru-so-draw-in-dwg (car so_file))
 );_ end of if
(ru-app-end)
 (princ)
);_ end of defun
(START)
```

Главная функция, рисующая спецификацию, приведена в листинге 32.6.

Листинг 32.6. Функция `ru-so-draw-in-dwg`

```
(defun ru-so-draw-in-dwg (so_file / lst_records block_ent first_block_ent
  left_top_pnt lst_columns_width number_page records)
  (if (findfile so_file)
;; Читаем файл спецификации
    (if (setq lst_records (ru-table-conv-csv-to-string-list so_file))
      (progn
        (setq number_page       1
;; Запрашиваем точку начала рисования
              left_top_pnt      (ru-get-point-reguired
                  "Верхний левый угол первого листа спецификации: "
                                 nil)
;; Список с размерами граф в миллиметрах на бумаге
              lst_columns_width
               (list 20.0 130.0 60.0 35.0 45.0 20.0 20.0 25.0 40.0)
;; Преобразуем список строк в список записей
              lst_records       (ru-table-conv-records-to-strings
                                  lst_records
                                  lst_columns_width
                                )
              records           (length lst_records)
       );_ end of setq
;; Пока не закончится список записей в спецификации
       (while (> records 0)
;; Рисуем требуемый бланк спецификации
         (setq block_ent  (ru-so-draw-forms left_top_pnt number_page)
;; пишем порцию строк, входящих на страницу, и получаем остаток строк
               lst_records  (ru-so-draw-records left_top_pnt number_page
                              lst_records lst_columns_width)
               left_top_pnt
       (polar left_top_pnt 0(ru-conv-millimeter-in-paper-to-unit 430.0))
               records      (length lst_records)
         );_ end of setq
;; заполняем основную надпись
         (ru-so-set-attribs block_ent number_page)
```

```
          (if (= number_page 1) (setq first_block_ent block_ent))
          (setq number_page (1+ number_page))
        );_ end of while
;; после рисования всех бланков
        (setq number_page (1- number_page))
;; заполняем графу "Листов" на первом листе
        (ru-block-change-attributes first_block_ent
          (list (cons "ALLPAGE" (itoa number_page)))
        );_ end of ru-block-change-attributes
  (princ (strcat "\nГотово. Начерчено " (itoa number_page) " листов \n"))
     );_ end of progn
      (ru-msg-alert "ОШИБКА: ru-table-conv-csv-to-string-list")
   );_ end of if
    (ru-msg-alert (strcat "Не найден \n" so_file))
 );_ end of if
)
```

Результат работы этой функции показан на рис. 32.6.

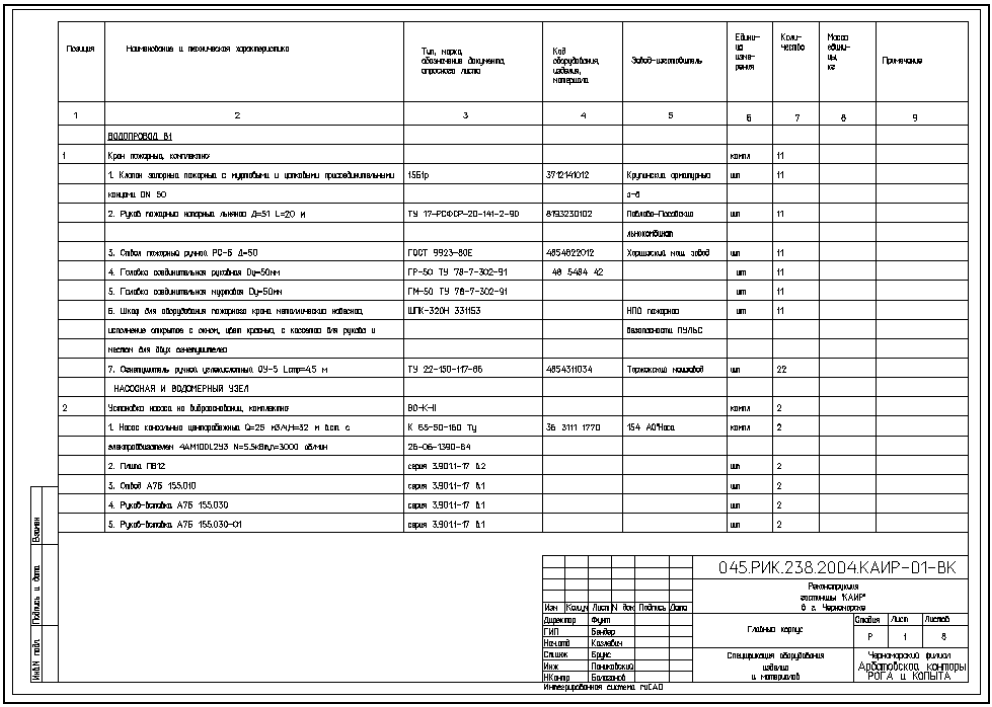


Рис. 32.6. Первый лист нарисованной спецификации оборудования

Наиболее значимые вспомогательные функции приведены в следующих четырех листингах (листинги 32.7—32.10).

Листинг 32.7. Функция ru-table-conv-csv-to-string-list

```
(defun ru-table-conv-csv-to-string-list
                                        (full_csv_file_name / result x)
;;; Преобразование csv в список
  (foreach string (cdr
                  (ru-list-read-from-file full_csv_file_name)
   )
    (setq result (cons
                  (mapcar '(lambda (x)
                         (ru-string-rem-all x (list "\"")))
                          (ru-string-to-list string ";")
                 );_ end of mapcar
                  result
               );_ end of cons
   );_ end of setq
 );_ end of foreach
  (reverse result)
);_ end of defun
```

Листинг 32.8. Функция ru-table-conv-records-to-strings

```
(defun ru-table-conv-records-to-strings (lst_records lst_columns_width /
column_index lst_field lst_fields_string lst_substr max_strings_in_row
                                        result)
;|-------------------------------------------------------------------
Преобразование списка длинных строк в список коротких строк с учетом переносов по
словам. Каждая запись выравнивается по высоте до максимального количества подстрок
-------------------------------------------------------------------|;
  (foreach record lst_records
    (setq max_strings_in_row  1
;; это присваивание нельзя убирать!!
       lst_fields_string '()
       column_index 0
   );_ end of setq
   ;;------------ разборка каждого поля -------------------------------
    (foreach field record
      (setq
        lst_field         (ru-string-division-into-syllables
                            field
                            (ru-conv-millimeter-in-paper-to-unit
                              (nth column_index lst_columns_width)
                            )
                          )
        column_index      (1+ column_index)
        lst_fields_string (cons lst_field lst_fields_string)
     );_ end of setq
   );_ end of foreach
    (setq result
          (append
            result
```

```
            (ru-list-Douglas-Wilson-transpose
              (ru-list-align-sublists-length
                  (reverse lst_fields_string) "")
            )
          );_ end of append
   );_ end of setq
 );_ end of foreach
  result
);_ end of defun
```

Листинг 32.9. Функция `ru-so-draw-forms`

```
(defun ru-so-draw-forms (ins_point number_page / block_ent format_block
 ins_scale)
  (if (= number_page 1)
    (setq format_block "ru-so-first")
    (setq format_block "ru-so-next")
 );_ end of if
  (if (not (tblsearch "block"  format_block))
   (setq format_block
     (strcat (ru-dirs-get-local-app-data) "so\\" format_block ".dwg"))
 )
  (setq ins_scale (ru-conv-millimeter-in-paper-to-unit 1.0)
        block_ent
                  (ru-block-insert-obj
                     format_block
                    ins_point
                    ins_scale
                    ins_scale
                    ins_scale
                    0
                  );_ end of ru-block-insert-obj
 );_ end of setq
  block_ent
);_ end of defun
```

Листинг 32.10. Функция `ru-so-draw-records`

```
(defun ru-so-draw-records (left_top_pnt number_page lst_records
  lst_columns_width / field fields number_rows num_column page pnt txt_pnt)
;; пишет тексты и удаляет написанное из списка
;; Количество строк на первом и последующих листах
  (if (= 1 number_page) (setq number_rows 21) (setq number_rows 26))
  (setq number_rows (min number_rows (length lst_records))
        pnt
;; Высчитываем начало текста в графе "Позиция"
 (polar left_top_pnt 0 (ru-conv-millimeter-in-paper-to-unit 22.0))
        pnt         (polar pnt
                           (ru-geom-go-right 0)
                           (ru-conv-millimeter-in-paper-to-unit 56.0)
                    );_ end of polar
```

```
        num_column  0
 );_ end of setq
  (repeat number_rows
    (setq
      num_column 0
      fields     (car lst_records)
      txt_pnt    pnt
   );_ end of setq
    (repeat (length fields)
      (setq field (nth num_column fields))
      (if (/= field "")
        (ru-text-draw field txt_pnt (ru-normal-text-height) 0.0 NIL)
     );_ end of if
      (setq txt_pnt    (polar txt_pnt
                              0
                              (ru-conv-millimeter-in-paper-to-unit
                                (nth num_column lst_columns_width)
                              )
                      );_ end of polar
            num_column (1+ num_column)
     );_ end of setq
   );_ end of repeat
;; Опускаемся на строку ниже
    (setq pnt (ru-geom-txt-down-line pnt));_ end of setq
    (setq lst_records (cdr lst_records))
 );_ end of repeat
  lst_records
);_ end of defun
```

Резюме

Используя материалы этой главы в качестве постановки задачи, любой программист баз данных сможет разработать рабочую программу — примеров кодирования мы не рассматривали специально.

Приведенные функции рисования спецификаций оборудования с помощью системы AutoCAD можно, с небольшими изменениями, использовать и для рисования "угловых" спецификаций с применением базы данных спецификаций оборудования.

ГЛАВА 33

Элементы документооборота

Без правильно организованного электронного (впрочем, как и без бумажного) документооборота невозможна эффективная работа проектных организаций[1]. Разумная организация работы с документами может дать больший эффект, чем автоматизация черчения. В большинстве проектных организаций, за некоторыми досадными исключениями, бумажные архивы проектов и иных документов (типовых проектов, серий, СНиП, паспортов оборудования) и системы работы с ними были организованы на достаточно высоком уровне.

К сожалению, при массовой компьютеризации про электронные архивы такого уже сказать нельзя. Электронный документооборот (не только в проектных организациях, но и в органах власти) находится в зачаточном состоянии. Руководители проектных организаций еще не поняли достоинств и недостатков современных информационных технологий — иначе они не пускали бы все "на самотек". По-прежнему "главным" считается бумажный оригинал, переданный в бумажный архив. Не вдаваясь в юридические проблемы с легитимностью электронных документов — они уже решаются на законодательном и практическом уровне, и в социально-экономические аспекты, мы рассмотрим элементарные технические решения, позволяющие пусть не устранить беспорядок, но дать возможность наведения порядка.

Разумеется, в рамках нашей САПР невозможно разработать настоящую систему электронного документооборота, но некоторые простейшие элементы, способствующие улучшению организации работы с документами, мы рассмотрим. В основном это будут задачи ведения и использования электронного архива чертежей.

Что требуется исполнителю

Любой исполнитель, работающий на компьютере, держит у себя собственный архив разработок, прихватывая еще все чужое, "что плохо лежит". А так лежит практически все.

---

[1] Напоминаем, что мы рассматриваем САПР для строительного проектирования, и совершенно не будем касаться организации документооборота в конструкторских организациях машиностроительного профиля, а также "канцелярского" документооборота и систем подготовки производства.

Исполнителю требуется не так уж много:

- иметь порядок в личном архиве;
- знать, что и где взять в доступном общем фонде.

Вспомним, зачем мы обычно обращаемся к традиционным архивам. Обычно требуется решить следующие задачи:

- посмотреть архивный экземпляр проекта во время разборок с заказчиками и строителями;
- подобрать подходящий проект для повторного применения;
- использовать готовый проект для разработки каких-то новых вариантов;
- повторно применить отдельные технические решения.

Изредка возникают и более сложные задачи — например, составление по готовым проектам каталогов и технико-экономических показателей для быстрой предварительной оценки потребности в ресурсах и сметной стоимости.

Исполнители заинтересованы во внедрении систем электронного документооборота и электронных архивов, прежде всего для наведения порядка на собственном компьютере. Многие, просто по незнанию, хранят файлы где попало, иногда не могут разыскать файл, сохраненный час назад. В *главе 3* мы уже рассуждали о размещении файлов и папок на компьютере пользователя, и предусмотрели некоторые средства для облегчения ведения собственного архива.

Электронный архив организации можно создавать по аналогичным принципам, но с учетом требований безопасности. Из общих архивов все хотят брать материалы, но не все хотят их пополнять.

При переходе на другую работу (иногда в конкурирующую фирму) все наработанные материалы разумный исполнитель непременно заберет с собой, а все, что было на прежнем месте — постарается уничтожить. Ведь это за ним "не числится". Его заставят разыскать и сдать давно утерянные книги, СНиП и сам компьютер, но никто и не вспомнит об *информационных ресурсах*, которые гораздо ценнее компьютера.

Что требуется руководителю

Многих руководителей сохранившихся (или бывших) государственных проектных организаций архивы интересуют только в виде площадей, которые можно было бы сдать в аренду. Электронный архив для них вначале интересен, как способ освободить место, но после того как становится ясно, что "загнать в компьютер" накопленные за десятилетия материалы быстро не удастся, интерес к электронному документообороту пропадает.

Руководители организаций "нового" типа, наоборот, действительно заинтересованы в электронном документообороте. Во многих таких фирмах работа с момента образования выполнялась на компьютерах, бумажные архивы еще небольшие и выполняют роль хранилища юридических документов. Реальная работа все чаще ведется с электронными документами.

Руководитель заинтересован в результатах работы всей организации, и в этом смысле его требования к архивам должны быть более высокими. Основные дополнительные требования заключаются в следующем:

- в электронный архив должны поступать все готовые проекты;
- доступ к электронному архиву должен производиться с учетом политики безопасности;
- все электронные проекты должны быть признаны информационными ресурсами, со всеми вытекающими последствиями.

Решить эти дополнительные задачи трудно — требуется подкрепление организационно-техническими мероприятиями. Разработать и утвердить стандарты предприятия по этим вопросам несложно, сложнее добиться их исполнения. Важнейшим для реализации должно быть простое условие — не "закрывать деньги" исполнителям, пока все материалы в требуемом виде не будут переданы в электронный архив.

Многие проектировщики не подозревают, что в соответствии с федеральным законодательством информационные ресурсы являются имуществом и должны ставиться на учет. Это требование практически нигде не выполняется, т. к. такое имущество возникает как бы ниоткуда (за исключением случаев покупки), а любое имущество должно быть оценено в денежном выражении и за него должны платиться налоги.

Что мы уже нечаянно сделали

Хотя мы не разрабатывали специально систему документооборота, у нас имеется много решений, способствующих ее внедрению в будущем:

- разработан стандарт приемов работы для строительного проектирования и все программы написаны так, чтобы обеспечить техническую поддержку его соблюдения;
- предусмотрена единообразная информация о свойствах каждого DWG-файла, а накопление такой информации позволит в будущем программно обрабатывать файлы и помещать сведения о них в базы данных системы документооборота;
- имеются функции извлечения свойств внешних файлов — соответственно, несложно сделать и функции поиска файлов по свойствам, в то числе в сочетании с имеющейся функцией поиска по атрибутам файла;
- предусмотрено ведение комментариев к папкам и файлам в "человеческом" виде, которые значительно облегчают разбирательства с файлами и могут быть использованы при формировании баз данных;
- разработан классификатор слоев, позволяющий единообразно именовать слои и устанавливать для них единые свойства;
- предусмотрены средства для проверки размещения объектов по слоям и для упорядочивания слоев;
- предусмотрено программное заполнение основных надписей по специальным стандартам, с сохранением данных в файлах для последующего учета в системе документооборота.

Конечно, этого мало, и необходима большая работа по созданию собственной системы документооборота или интеграции с покупными системами. Пока мы разработаем еще несколько дополнительных решений.

Организация архива электронных калек

В электронном архиве можно и нужно хранить готовые проекты "как есть", сжатые специальными программами-архиваторами. В такой архив должны включаться не только DWG-файлы, но и вспомогательные — файлы комментариев, основных надписей и т. п. Но пользоваться такими материалами не очень удобно. По нашему опыту использования геоинформационных технологий *(см. главу 34)* очень удобно использование так называемых электронных калек.

Электронная калька — это отдельный слой, сохраненный в виде файла в один из подкаталогов специальной папки, размещенной, как правило, на общедоступном диске (рис. 33.1). Непременным условием для всех электронных калек одного проекта является расположение точки вставки в начале мировой системы координат, при этом с началом МСК должно совпадать характерное место изображения в пространстве модели (пересечение осей, угол здания и т. п.). Для топографических планов и основанных на них рисунках начало мировой системы координат совпадает с местной системой координат населенного пункта.

При работе с кальками-слоями выполняются следующие операции:

- запись выбранных классифицированных слоев в соответствующую папку архива;
- выбор из архива калек-слоев и вставка в активный документ в виде внешних ссылок (всегда в пространство модели, в точку с координатами 0, 0 МСК, на слой, соответствующий слою кальки);

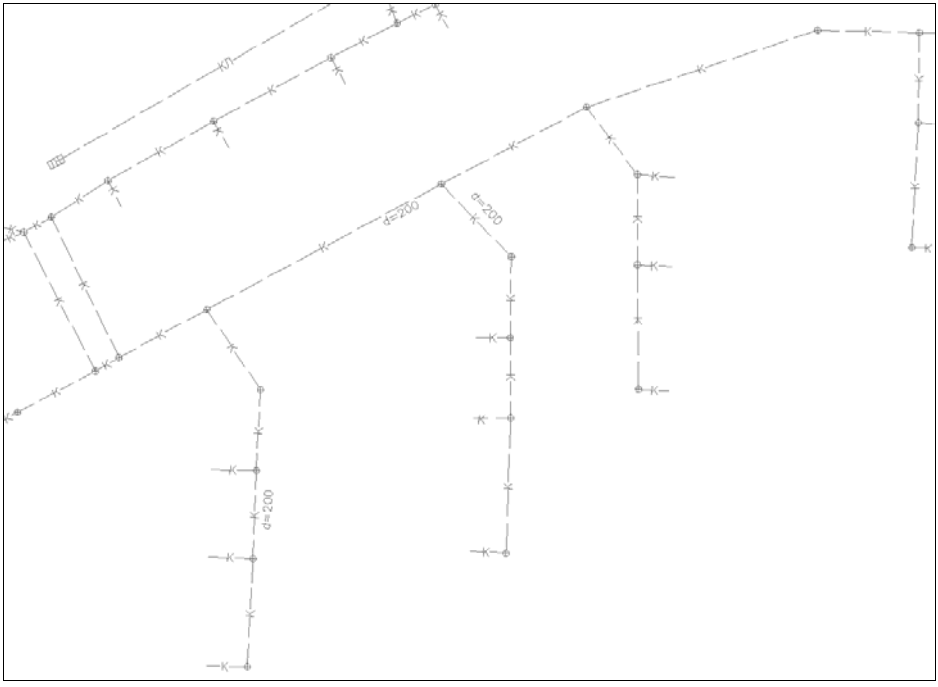


Рис. 33.1. Пример слоя-кальки с планом канализации

- сохранение, редактирование и загрузка именованных наборов калек, обеспечивающих быстрое формирование изображения из слоев, вставляемых в требуемом порядке (рис. 33.2).

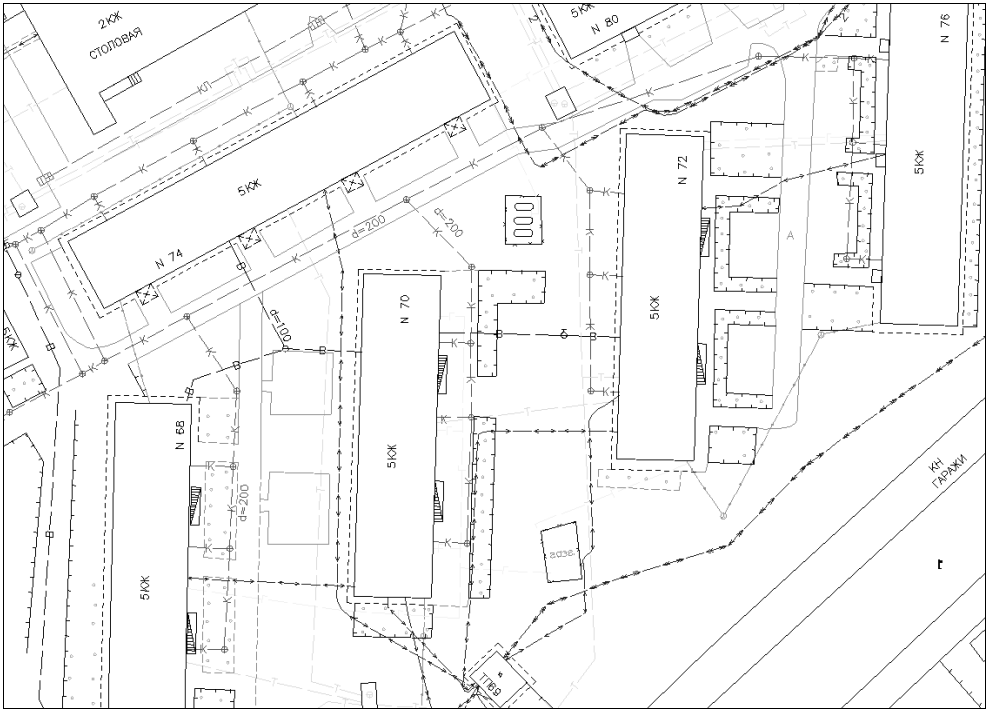


Рис. 33.2. Пример набора калек

Более подробно эти операции будут рассмотрены далее при разработке функций для работы с архивом.

Организация архива типовых проектных решений

Электронный архив *типовых проектных решений* (ТПР) предназначен для хранения любых рисунков, которые можно использовать повторно. По сути, это просто упорядоченный и комментированный набор файлов, которые можно вставлять в виде блоков. Но такой архив не должен превращаться и в обычную свалку. Попадать в него должны рисунки высокого качества, разбитые по темам и вытащенные из реальных проектов. Например, в ТПР не стоит помещать чертежи здания со всеми слоями, этажами, планами, разрезами, но можно поместить планы типовых секций различных серий (каждый этаж отдельно, рис. 33.3), компоновки насосных, тепловых пунктов, различные узлы (рис. 33.4).

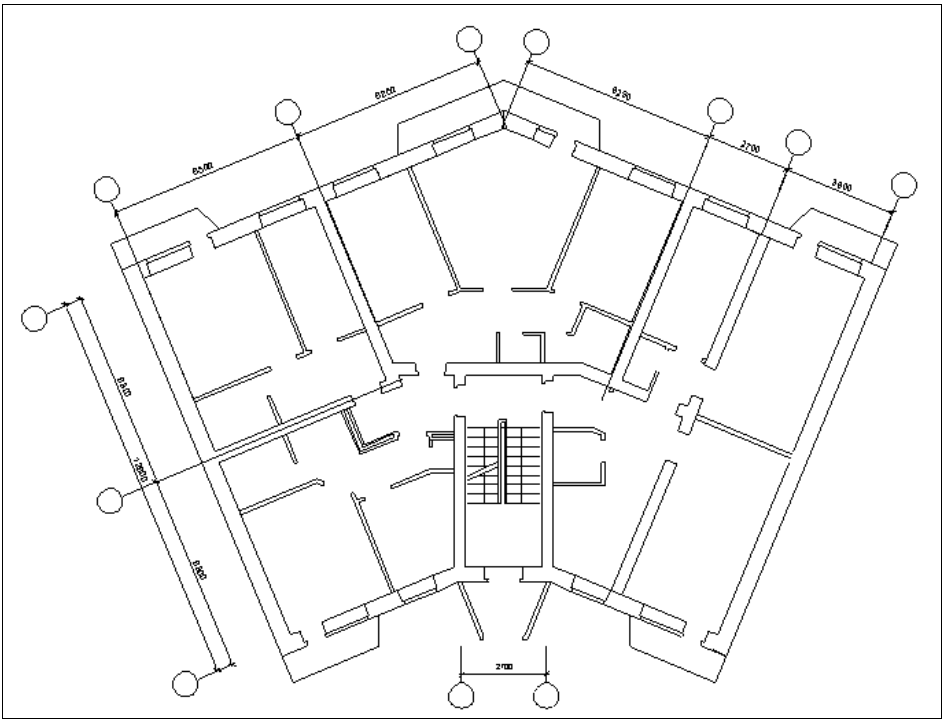


Рис. 33.3. Типовое проектное решение плана этажа

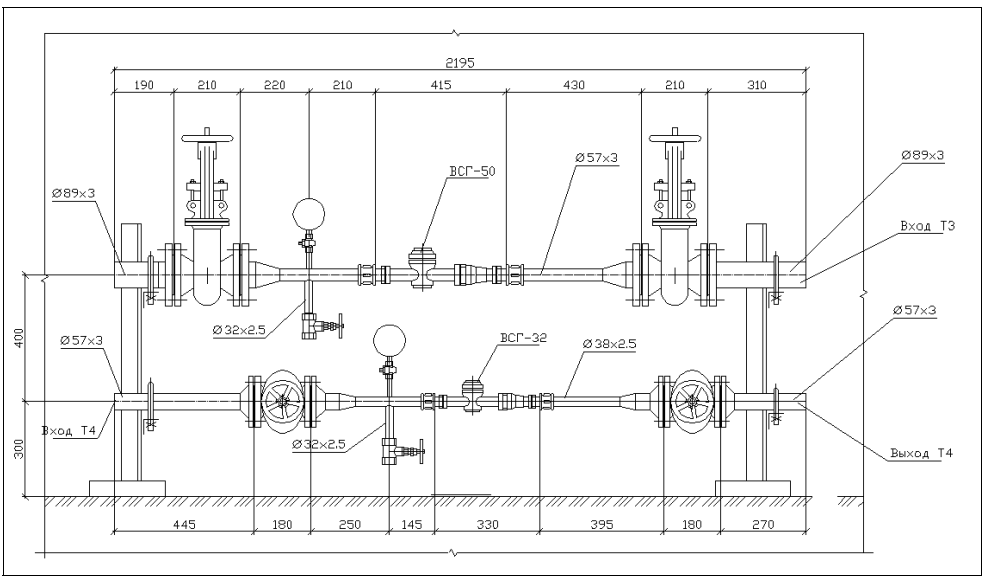


Рис. 33.4. Типовое проектное решение водомерного узла

Формирование архива ТПР должно производиться с учетом технической политики, проводимой в организации. Типовое решение, применяемое в одной организации, может оказаться предметом критики в другой.

Файл, помещаемый в архив ТПР, также должен соответствовать следующим необходимым требованиям:

- обязательно сопровождаться комментариями;
- иметь словесное описание точки вставки;
- иметь описание единиц рисунка для автоматического масштабирования при вставке;
- по возможности не включать масштабируемые символы и размеры.

Основные операции при работе с архивом ТПР сводятся к помещению файлов в архив и выбору из архива со вставкой в виде блока и будут рассмотрены далее.

Архив топографических планшетов

Такие архивы используются в немногих проектных организациях, использующих геоинформационные технологии, поэтому мы дадим о них только краткую справку.

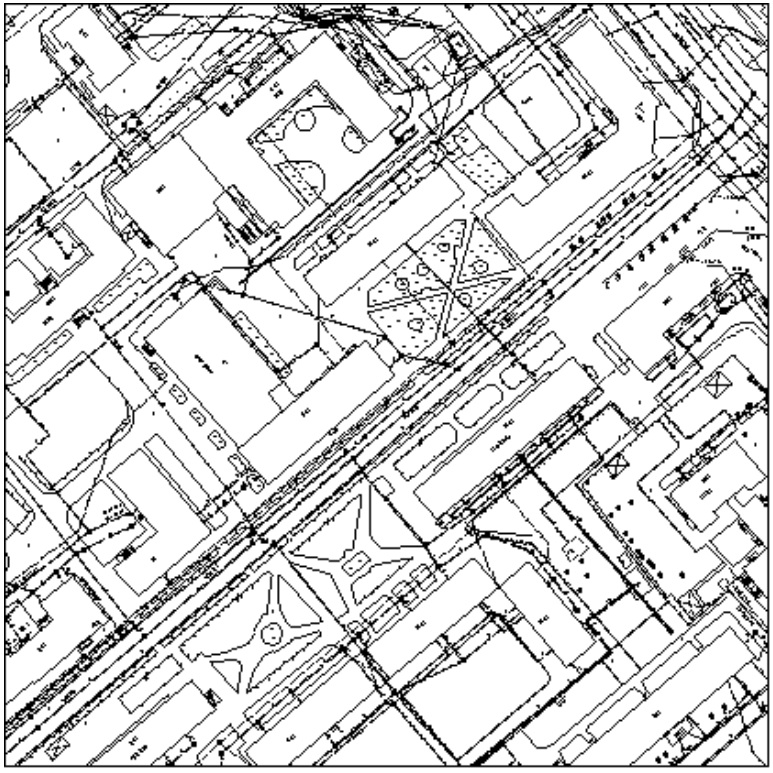


Рис. 33.5. Электронный топографический планшет

В архиве планшетов содержатся файлы электронных *топографических планшетов*. Каждый файл размещается в каталоге со строго определенным именем, и сам имеет имя, формируемое по определенному алгоритму. Правила именования каталогов и файлов зависят от принятой в населенном пункте *системы разграфки* плана местности (существует несколько известных нам систем).

Каждый электронный планшет (рис. 33.5) отображает квадратный фрагмент плана местности определенного размера (например, планшет масштаба 1:500 — 250×250 м), включает весь набор классифицированных слоев и имеет точку вставки, совпадающую с началом местной системы координат территории.

Планшеты могут быть нарисованы с использованием средств топографического рисования ruCAD, но могут включать и растровые подложки. Технологии создания планшетов можно посвятить целую книгу. В архив планшеты попадают после тщательного контроля в индивидуальном порядке.

Работа проектировщика, имеющего доступ к архиву планшетов, заключается в формировании сводного плана несколькими способами:

- указанием точки на плане;
- указанием прямоугольника;
- указанием известных номеров планшетов.

При любом способе в рисунок вставляются в нужное место требуемые планшеты, обнаруженные в архиве.

Программ для работы с планшетами в этой книге мы приводить не будем.

Функции для работы с электронными архивами

Для работы с электронными архивами нам потребуются несколько функций. Начнем с электронных калек-слоев.

Замечание

Все функции для работы с электронными архивами, помимо блокировки на уровне меню, проверяют права пользователя на соответствующие действия.

Как записать кальки в архив

Для записи калек напишем соответствующую программу (листинг 33.1).

Листинг 33.1. Файл ru_doc_href_layer_save.lsp

```
(defun start (/ _layers_list _dialog_list _selected_list _enabled_list
              _comments_list _save_layer _select_layers _make_lists
              folder folder_comment i)
;;-----Локальная функция ----------------------------------------
  (defun _make_lists (/ comment)
```

```
;|
Составляет списки для вывода в диалоговое окно. Допускается отметка только
классифицированных слоев
|;
    (setq _layers_list (ru-obj-list-layers))
    (foreach layer _layers_list
      (if (ru-layer-is-standard layer)
        (setq _enabled_list  (cons t _enabled_list)
              _selected_list (cons t _selected_list)
              comment        (ru-layer-read-comment layer)
              _comments_list (cons comment _comments_list)
       );_ end of setq
        (setq _enabled_list  (cons NIL _enabled_list)
              _selected_list (cons NIL _selected_list)
              comment        "Нет в классификаторе"
              _comments_list (cons comment _comments_list)
       );_ end of setq
     );_ end of if
;; Комментарии для отображения
      (setq
        _dialog_list
         (cons
           (if (= comment "")
             layer
             (strcat layer " [" comment "]")
         );_ end of if
          _dialog_list
       );_ end of cons
     );_ end of setq
   );_ end of foreach
    (setq _enabled_list  (reverse _enabled_list)
          _dialog_list   (reverse _dialog_list)
          _comments_list (reverse _comments_list)
          _selected_list (reverse _selected_list)
   );_ end of setq
 );_ end of defun
;;-----Локальная функция выбора слоев в диалоге-------------------
  (defun _select_layers ()
    (setq _selected_list
          (ru-dlg-show-check-list
            "Отбор калек слоев для записи в архив"
            _dialog_list
            _selected_list
            _enabled_list
         );_ end of ru-dlg-show-check-list
   );_ end of ru-dlg-user-show-check-list
 );_ end of defun
;;-----Локальная функция сохранения слоя в файл-------------------
  (defun _save_layer (layer comment dir / selection file_name
                                          file_info_list)
```

```
    (if (and
          (setq selection (ssget "_X" (list (cons 8 layer))))
;; Внешние ссылки не сохраняем
          (setq selection (ru-ss-exclude-xref selection))
       );_ end of and
      (progn
        (setq file_name
              (strcat (ru-dirs-path-and-slash dir)
                      layer
                      ".dwg"
             );_ end of strcat
       );_ end of setq
        (if (findfile file_name)
          (progn
            (setq file_info_list (dos_file file_name))
            (if (not (ru-no
                       (strcat
                         "Файл \n"
                         (nth 0 file_info_list)
                         (if (/= comment "")
                           (strcat "\n ["
                                   comment
                                   "]"
                            );_ end of strcat
                          );_ end of if
                         "\nСоздан "
                         (nth 2 file_info_list)
                         " в "
                         (nth 3 file_info_list)
                         ", размер "
                         (nth 1 file_info_list)
                         " байт"
                         "\nУже существует!"
                         "\nПереписать"
                       );_ end of strcat
                     );_ end of
             );_ end of not
             (if (dos_openp file_name)
               (ru-msg-alert
                 (strcat
                   "\nФайл "
                   file_name
                   "\nсейчас нельзя переписать, он открыт!"
                );_ end of strcat
              );_ end of ru-msg-alert
               (progn
                 (princ (strcat "\nЗаписываю " file_name "\n"))
                 (command "_.WBLOCK" file_name "_Y" "" "0,0"
                          selection "");_ end of command
```

```
                (ru-file-write-dirinfo-file-comment file_name comment)
                (princ "\nOK\n")
              );_ end of progn
            );_ end of if
          );_ end of if
        );_ end of progn
         (progn
           (princ (strcat "\nЗаписываю " file_name "\n"))
           (command "_.WBLOCK" file_name "" "0,0"
                    selection "")
           (ru-file-write-dirinfo-file-comment file_name comment)
           (princ "\nOK\n")
        );_ end of progn
      );_ end of if
    );_ end of progn
     (princ
       (strcat "\nСлой " layer
         " не содержит примитивов входящих в кальки"
      );_ end of strcat
    );_ end of princ
  );_ end of if
 );_ end of defun
;;-----Главная функция ----------------------------------------
  (ru-app-begin)
  (if (and (ru-user-action-enabled "PUT_LAY" t)
           (setq folder (ru-dlg-archive-layers-folder))
           (setq folder_comment
               (ru-dirs-read-dirinfo-dir-comment folder))
    );_ end of and
    (if (ru-yes
          (strcat  "\nОткрыта ПАПКА: " folder_comment
            "\nКаталог папки: " folder
            "\nЖелаете записать кальки в эту папку"
         );_ end of strcat
      );_ end of
      (progn
        (ru-user-write-last-param
          "archive_layers_folder"
          folder
       );_ end of ru-user-write-last-param
;; Создаем списки слоев для вывода в диалоговом окне
        (_make_lists)
;; Выводим диалог для выбора сохраняемых слоев
        (if (_select_layers)
          (progn
;; Устанавливаем мировую систему координат
            (command "_.UCS" "_W")
            (setq i 0)
;; Сохраняем отмеченные слои
            (foreach layer _layers_list
```

```
                (if
                  (nth i _selected_list)
                   (_save_layer layer (nth i _comments_list) folder)
                );_ end of if
                 (setq i (1+ i))
              );_ end of foreach
;; Восстанавливаем систему координат
               (command "_.UCS" "_P")
            );_ end of progn
          );_ end of if
        );_ end of progn
      );_ end of if
    );_ end of if
  (ru-app-end)
  (princ)
);_ end of defun
(start)
```

Эта программа выводит диалоговое окно выбора архивной папки для сохранения калек (рис. 33.6), затем, после подтверждения записи калек в эту папку (рис. 33.7), выводит список слоев (рис. 33.8), в котором доступны для отметки только стандартные классифицированные слои. После отметки экспортируемых слоев кальки записываются в архив.

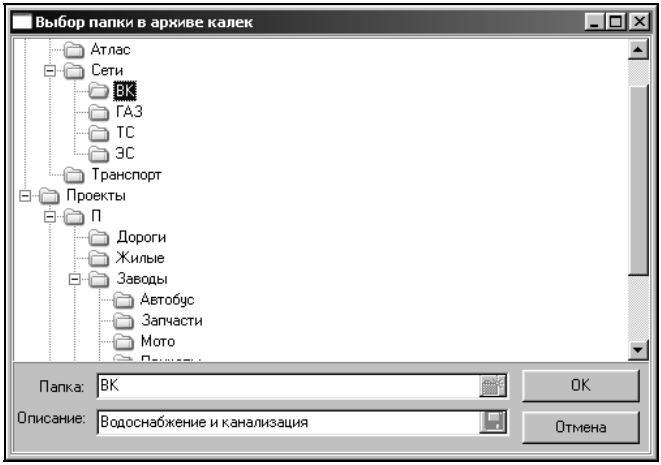


Рис. 33.6. Выбор папки в архиве калек

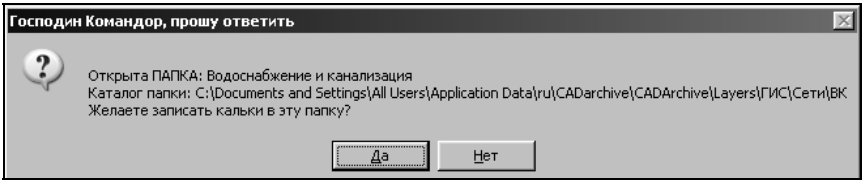


Рис. 33.7. Подтверждение выбора папки

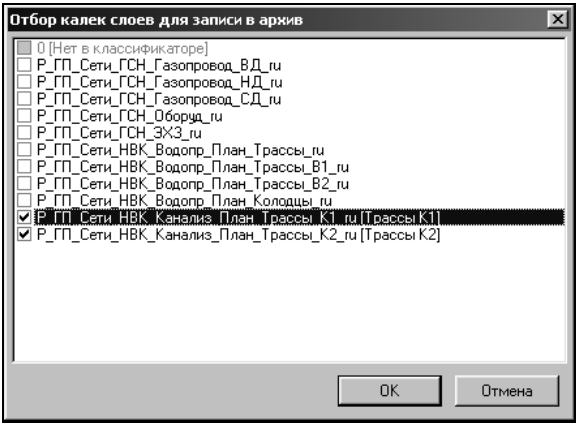


Рис. 33.8. Отбор слоев для записи в виде калек

Как взять кальки из архива

Для вставки одной или нескольких калек из архива напишем программу (листинг 33.2), вставляющую отобранные кальки в виде внешних ссылок.

Листинг 33.2. Файл ru_doc_href_layer_insert.lsp

```
(defun START (/ files_list)
  (ru-app-begin)
  (if (ru-user-action-enabled "GET_LAY" t)
    (if
      (setq files_list
             (ru-dlg-file-select-dwg-multi
               "Выбор калек" (ru-dirs-archive-layers-dir)
               (ru-user-read-last-param
                 "archive_layers_folder"
                 (ru-dirs-archive-layers-dir)
              );_ end of ru-user-read-last-param
               (ru-user-action-enabled "PUT_LAY" nil)
            );_ end of ru-file-dwg-multi-select
    );_ end of setq
      (foreach fn files_list (ru-block-insert-xref-from-archive fn))
   );_ end of if
 );_ end of if
  (ru-app-end)
  (princ)
);_ end of defun
(START)
```

Вставка каждой кальки производится с помощью функции, приведенной в листинге 33.3.

Листинг 33.3. Функция ru-block-insert-xref-from-archive

```
(defun ru-block-insert-xref-from-archive (full_name /
xref_name comment)
  (if (findfile full_name)
    (progn
      (setq comment
             (ru-file-read-dirinfo-file-comment full_name)
     );_ end of setq
      (princ (strcat "\nПодождите, вставляю \n "
                     full_name
                     (if (/= comment "")
                       (strcat "\n ["
                               comment
                               "]"
                      );_ end of strcat
                       ""
                    );_ end of if
                     "\n"
             );_ end of strcat
     );_ end of princ
      (setq xref_name (ru-file-unique-name full_name))
      (ru-xref-detach xref_name)
      (ru-xref-attach full_name xref_name)
   );_ end of progn
    (princ (strcat "\nНе найдена калька " full_name "\n"))
 );_ end of if
  xref_name
);_ end of defun
```

У нас может оказаться несколько калек одноименных слоев, но вставляемых из разных папок. Для предупреждения конфликтов разработана специальная функция (листинг 33.4), генерирующая по полному имени файла уникальную строку, используемую при вставке и отображаемую в стандартном диалоговом окне **Xref Manager** (Диспетчер внешних ссылок) как **Reference Name** (Имя).

Листинг 33.4. Функция ru-file-unique-name

```
(defun ru-file-unique-name (file_name /  i code)
  (setq i 0  code 0)
  (repeat (strlen file_name)
    (setq i (1+ i))
    (setq code (+ code (* (ascii (substr file_name i 1)) i)))
 );_ end of repeat
  (itoa code)
)
```

Вставка внешней ссылки производится всегда в пространство модели при установленной мировой системе координат (листинг 33.5).

Листинг 33.5. Функция ru-xref-attach

```
(defun ru-xref-attach (file_name ref_name / old_layer)
  (setq old_layer (getvar "CLAYER"))
  (command "_.UCS" "_W")
  (ru-layer-current (nth 2 (dos_splitpath file_name)))
  (vla-attachexternalreference
    (ru-obj-get-model-space) file_name ref_name
    (vlax-3d-point '(0.0 0.0 0.0)) 1.0 1.0 1.0 0.0 :vlax-false
 );_ end of vla-AttachExternalReference
  (command "_.UCS" "_P")
  (ru-layer-current old_layer)
);_ end of defun
```

Отсоединение внешней ссылки осуществляется функцией `ru-xref-detach` (листинг 33.6), предусматривающей предотвращение ошибок при попытке отсоединения несуществующей ссылки.

Листинг 33.6. Функция ru-xref-detach

```
(defun ru-xref-detach (name)
;| Примеры:
Несуществующая
(ru-xref-detach "1133177") вернет nil
Существующая
(ru-xref-detach "1406897") вернет T
|;
  (if (vl-catch-all-error-p
        (vl-catch-all-apply
          '(lambda ()
             (vla-detach
               (vla-item (ru-obj-get-blocks)
                         name
              );_ end of vla-item
            );_ end of vla-detach
          );_ end of lambda
        );_ end of setq
     );_ end of vl-catch-all-error-p
    nil
    T
 );_ end of if
);_ end of defun
```

Как использовать наборы калек

Сохранение набора калек производится программой, приведенной в листинге 33.7.

Листинг 33.7. Файл ru_doc_href_layer_set_save.lsp

```
(defun START (/ archive_layers_dir file file_name set_file_name
              xref_list xref_list2)
```

```
  (if (ru-user-action-enabled "PUT_LAY" t)
    (if (setq xref_list (ru-xref-list-with-path))
      (if (setq xref_list2
;|
Используем диалоговое окно с двойным списком, в котором можно не только
сформировать результирующий список, но и изменить порядок его элементов,
т. е. в какой последовательности слои будут впоследствии вставляться в рисунок
|;
                (ru-dlg-file-dual-select-by-note
                  "Отбор внешних ссылок для сохранения в наборе"
                  "Имеются в рисунке"
                  "Поместить в набор"
                  xref_list
                );_ end of ic_SelectList
        );_ end of setq
        (if (setq set_file_name
                  (_ru-dlg-file
                    "Выбор или создание набора калек"
                    (setq archive_layers_dir
                     (ru-dirs-archive-layers-dir))
                    (ru-user-read-last-param
                      "archive_layers_folder"
                      (ru-dirs-archive-layers-dir)
                    );_ end of ru-user-read-last-param
                    "*.rulset" "" nil T T
                  );_ end of _ru-dlg-file
          );_ end of setq
          (progn
            (setq set_file_name (car set_file_name))
            (if (setq file (open set_file_name "w"))
              (progn
               ;; пишем список ссылок в файл
                (foreach file_name xref_list2
                  (princ
                    (strcat
;; Преобразуем полное имя в относительное
                      (ru-string-trim-prefix archive_layers_dir
                       (car file_name)) "\n"
                    );_ end of strcat
                    file
                  );_ end of princ
                );_ end of foreach
                (setq file (close file))
                (princ "\nГотово!")
              );_ end of progn
              (ru-msg-alert (strcat "Нельзя записать файл\n"
                                    set_file_name))
            );_ end of if
          );_ end of progn
          (ru-msg-alert "Файл набора не выбран!")
        );_ end of if
```

```
        (ru-msg-alert "Не выбран список сохраняемых ссылок!")
     );_ end of if
      (ru-msg-alert "Нет внешних ссылок для сохранения!")
   );_ end of if
 );_ end of if
  (princ)
);_ end of defun
(start)
```

Открыть типовой набор калек можно с помощью программы, приведенной в листинге 33.8.

Листинг 33.8. Файл ru_doc_href_layer_set_restore.lsp

```
(defun START (/ archive_layers_dir file files_list s set_file_name)
  (if (ru-user-action-enabled "GET_LAY" T)
    (progn
      (if (setq set_file_name
                (_ru-dlg-file
                  "Выбор  набора калек"
                  (setq archive_layers_dir (ru-dirs-archive-layers-dir))
                  (ru-user-read-last-param "archive_layers_folder"
                    (ru-dirs-archive-layers-dir)
                  );_ end of ru-user-read-last-param
                  "*.rulset" "" nil
                  (ru-user-action-enabled "PUT_LAY" nil)
                  nil
                );_ end of _ru-dlg-file
        );_ end of setq
        (progn
          (setq set_file_name (car set_file_name))
          (if (setq file (open set_file_name "r"))
            (progn
              (while (setq s (read-line file))
                (if (/= s "")
                  (progn
                    (setq s (ru-string-trim-left s))
                    (if
                      (/= (substr s 1 1) ";")
                      ;; пропускаем закомментированные
                       (setq files_list (cons S files_list))
                    );_ end of if
                  );_ end of progn
                );_ end of if
              );_ end of while
              (close file)
              ;; Реверс для вставки в обратном порядке
              (foreach fn (reverse files_list)
                (ru-block-insert-xref-from-archive
                  (strcat archive_layers_dir fn)
```

```
              );_ end of ru-block-insert-xref-from-archive
             );_ end of foreach
              (princ "\nГотово!")
           );_ end of progn
            (ru-msg-alert (strcat "Нельзя прочитать файл\n"
                 set_file_name))
        );_ end of if
       );_ end of progn
     );_ end of if
   );_ end of progn
 );_ end of if
  (princ)
);_ end of defun
(START)
```

Отредактировать, при необходимости, набор калек можно с помощью программы, приведенной в листинге 33.9.

Листинг 33.9. Файл ru_doc_href_layer_set_edit.lsp

```
(defun START (/ archive_layers_dir file files_list s set_file_name
               xref_list2)
  (if (ru-user-action-enabled "PUT_LAY" T)
    (progn
      (if (setq set_file_name
                (_ru-dlg-file
                  "Выбор  набора калек для редактирования"
                  (setq archive_layers_dir (ru-dirs-archive-layers-dir))
                  (ru-user-read-last-param
                    "archive_layers_folder"
                     (ru-dirs-archive-layers-dir)
                   );_ end of ru-user-read-last-param
                  "*.rulset" "" nil
                  (ru-user-action-enabled "PUT_LAY" nil)
                  nil
                );_ end of _ru-dlg-file
        );_ end of setq
        (progn
          (setq set_file_name (car set_file_name))
          (if (setq file (open set_file_name "r"))
            (progn
              (while (setq s (read-line file))
                (if (/= s "")
                  (progn
                    (setq s (ru-string-trim-left s))
                    (if
                      (/= (substr s 1 1) ";")
                      ;; пропускаем закомментированные
                       (setq files_list
                              (cons (strcat archive_layers_dir S)
```

```
                                files_list
                            );_ end of cons
                        );_ end of setq
                    );_ end of if
                );_ end of progn
            );_ end of if
        );_ end of while
        (close file)
        ;; Реверс для вставки в обратном порядке
        (if (setq xref_list2
                  (ru-dlg-file-dual-select-by-note
                    "Отбор калек для сохранения в типовом наборе"
                    "Уже имеются в наборе"
                    "Оставить в наборе"
                    files_list
                  );_ end of ru-dlg-file-dual-select-by-note
            );_ end of setq
          (if (setq file (open set_file_name "w"))
            (foreach file_name xref_list2
              (princ
                (strcat
                  (ru-string-trim-prefix
                    archive_layers_dir
                    (car file_name)
                  );_ end of ru-string-trim-prefix
                  "\n"
                );_ end of strcat
                file
              );_ end of princ
            );_ end of foreach
            (ru-msg-alert (strcat "Нельзя записать файл\n"
                          set_file_name))
          );_ end of if
        );_ end of if
      );_ end of progn
      (ru-msg-alert (strcat "Нельзя прочитать файл\n"
      set_file_name))
    );_ end of if
  );_ end of progn
 );_ end of if
);_ end of if
  (princ)
);_ end of defun
(START)
```

Присоединить кальку к рисунку можно с помощью программы, приведенной в листинге 33.10, а удалить ("вернуть в архив") — с помощью программы, приведенной в листинге 33.11.

Листинг 33.10. Файл ru_doc_href_insert.lsp

```
(defun START (/ comment ent file_name layer xref_name)
  (ru-app-begin)
  (if
    (not
      (ru-no
        (strcat
          "ВНИМАНИЕ!"
          "\nДобавление кальки сделает ее постоянной частью рисунка!"
          "\nКалька будет добавлена в расчлененном виде,"
          "\nс возможностью редактирования входящих в нее примитивов!"
          "\nРазмер рисунка увеличится, а слой кальки НЕ БУДЕТ обновляться!"
          "\nДействительно хотите добавить кальки"
       );_ end of strcat
     );_ end of ru-msg-no
   );_ end of not
     (progn
       (while (setq ent (ru-get-entsel-by-type
                          "Выбери добавляемую кальку"
                          "Это не КАЛЬКА"
                          (list "INSERT")
                          T
                        );_ end of ru-get-entsel-by-type
           );_ end of setq
         (setq ent (car ent))
         (if (ru-block-is-xref ent)
           (progn
             (setq
               xref_name (cdr (assoc 2 (entget ent)))
               file_name (cdr (assoc 1 (tblsearch "BLOCK" xref_name)))
               comment   (ru-file-read-dirinfo-file-comment file_name)
               layer     (ru-xref-short-layer-name xref_name)
            );_ end of setq
             (redraw ent 3)
             (if
               (ru-yes (strcat "Добавить кальку\n"
                               comment
                               "\n " file_name
                               "\n в слой " layer
                       );_ end of strcat
             );_ end of ru-yes
               (progn
                 (redraw ent 4)
                 (ru-xref-detach xref_name)
                 (command "_.INSERT"
                          (strcat "*" file_name)
                          (list 0.0 0.0) "1" "0"
                  );_ end of command
                  (princ "\nГотово!")
              );_ end of progn
```

```
                (redraw ent 4)
              );_ end of if
            );_ end of progn
             (princ "\nЭто НЕ КАЛЬКА!")
          );_ end of if
        );_ end of while
      );_ end of progn
  );_ end of if
   (ru-app-end)
   (princ)
);_ end of defun
(START)
```

Листинг 33.11. Файл ru_doc_href_delete.lsp

```
 (defun START (/ comment ent file_name xref_name)
  (while (setq ent (ru-get-entsel-by-type
                     "Выбери возвращаемую кальку"
                     "Это не КАЛЬКА"
                     (list "INSERT")
                     T
                   );_ end of ru-get-entsel-by-type
         );_ end of setq
    (setq ent (car ent))
    (if (ru-block-is-xref ent)
      (progn
        (setq
          xref_name (cdr (assoc 2 (entget ent)))
          file_name (cdr (assoc 1 (tblsearch "BLOCK" xref_name)))
          comment   (ru-file-read-dirinfo-file-comment file_name)
        );_ end of setq
        (redraw ent 3)
        (if
          (ru-yes (strcat "Вернуть кальку\n" comment "\nв " file_name))
           (progn
             (redraw ent 4)
             (ru-xref-detach xref_name)
             (princ "\nГотово!")
           );_ end of progn
           (redraw ent 4)
        );_ end of if
     );_ end of progn
      (princ "\nЭто НЕ КАЛЬКА!")
   );_ end of if
 );_ end of while
  (princ)
);_ end of defun
(START)
```

Кроме приведенных программ, возможно ведение списка "любимых" наборов и калек, но рассматривать эти программы мы не будем — реализация их очевидна.

Работа с типовыми проектными решениями

Запись типового проекта в специальный каталог производится с помощью программы, приведенной в листинге 33.12.

Листинг 33.12. Файл ru_doc_project_save.lsp

```
(defun start (/ lst projects_dir file_name comment)
  (if (ru-user-action-enabled "PUT_PROJ" T)
    (if (ru-yes (strcat "\ВНИМАНИЕ!"
          "\nТиповой проект предназначен для повторного использования."
          "\nЗаписывается в специальную папку архива,"
          "\nсопровождается комментарием."
          "\nБудем делать типовой проект"
       );_ end of strcat
     );_ end of ru-msg-yes
       (progn
         (setq projects_dir (ru-dirs-archive-projects-dir))
         (if (setq lst (ru-block-write-block
                         "Имя файла для записи типового проекта"
                         (ru-user-read-last-param "LastProjectsDir"
                          projects_dir)
                          projects_dir
                       )
           );_ end of setq
           (progn
             (setq file_name (car lst) comment (cadr lst))
             (ru-user-write-last-param
               "LastProjectsDir"
               (ru-file-path file_name)
             );_ end of ru-user-write-last-param
             (ru-dwgprops-file-edit
               file_name
               (list (cons 300 (rtos (ru-scale-model)))
                     (cons 301 (ru-unit-name))
              );_ end of list
             );_ end of ru-dwgprops-file-edit
             ;; Загрузить записанный файл для просмотра и правки
             (if
               (ru-yes (strcat "\nЗаписан типовой проект "
                         file_name "\nс комментарием '"
                         comment
                         "'\n\nЗагрузить его для просмотра?"
                );_ end of strcat
              );_ end of ru-msg-yes
;|Открываем записанный файл в другом окне. |;
                (ru-dwg-open-file file_name)
             );_ end of if
           );_ end of progn
         );_ end of if
```

```
      );_ end of progn
   );_ end of if
 );_ end of if
  (princ)
);_ end of defun
(START)
```

Непосредственная запись файла производится с помощью функции `ru-block-write-block` (листинг 33.13). Эта функция записывает блок в файл, выбираемый или создаваемый в заданном каталоге.

Листинг 33.13. Функция `ru-block-write-block`

```
(defun ru-block-write-block (caption start_dir root_dir / base_point
comment file_name note_base_point selection)
;; возвращает список из имени записанного файла и комментария
  (princ "\nЖдите, загружаю навигатор...")
  (if
    (setq file_name
           (ru-dlg-file-select-or-new-dwg caption start_dir root_dir
             "" T)
   );_ end of setq
     (progn
       (setq file_name (car file_name)
             comment   (ru-file-read-dirinfo-file-comment file_name)
     );_ end of setq
       (princ "\nВыбор объектов для записи:\n")
       (if (setq selection (ru-ss-get-or-ssfirst))
         (progn
           (setq base_point  (ru-get-point-reguired
                       "Базовая точка для последующих вставок"
                             nil)
         );_ end of setq
           (ru-block-write-with-comment file_name comment base_point
             selection T)
       );_ end of progn
     );_ end of if
       (list file_name comment)
    );_ end of progn
     nil
 );_ end of if
);_ end of defun
```

В листингах 33.14 и 33.15 приведены функции, позволяющие записывать или перезаписывать блоки в файлы с комментариями.

Листинг 33.14. Функция `ru-block-write-with-comment`

```
(defun ru-block-write-with-comment (file_name comment base_point
   selection is_ask_rewrite / file_info_list rewrite)
```

```
  (if (and (findfile file_name) is_ask_rewrite)
    (setq
      file_info_list
                    (dos_file file_name)
      rewrite       (not (ru-no (strcat "Файл \n"
                                (nth 0 file_info_list)
                                (if (/= comment "")
                                  (strcat "\n ["
                                          comment
                                          "]"
                                  );_ end of strcat
                                );_ end of if
                                "\nСоздан " (nth 2 file_info_list)
                                " в " (nth 3 file_info_list)
                                ", размер " (nth 1 file_info_list)
                                " байт" "\nУже существует!"
                                "\nПереписать"
                            );_ end of strcat
                        );_ end of
                    );_ end of not
   );_ end of setq
    (setq rewrite T)
 );_ end of if
  (if rewrite
    (ru-block-rewrite-with-comment file_name comment base_point
                                                    selection)
 );_ end of if
);_ end of defun
```

Листинг 33.15. Функция `ru-block-rewrite-with-comment`

```
(defun ru-block-rewrite-with-comment (file_name comment base_point
                                      selection)
  (if (dos_openp file_name)
    (progn
      (ru-msg-alert (strcat "\nФайл " file_name
          "\nсейчас нельзя переписать, он открыт!"
       ))
      nil
   );_ end of progn
    (progn
      (princ (strcat "\nЗаписываю " file_name
                     (if (/= comment "") (strcat "\n [" comment "]"))
                     "\n"))
      (command "_.WBLOCK" file_name)
      (if (findfile file_name) (command "_Y"))
      (command "" base_point selection "")
      (ru-file-write-dirinfo-file-comment file_name comment)
      t
```

```
   );_ end of progn
 );_ end of if
);_ end of defun
```

Функция открытия файла (листинг 33.16) пригодится нам не только для типовых проектов.

Листинг 33.16. Функция `ru-dwg-open-file`

```
(defun ru-dwg-open-file (dwg_file)
;| Открывает DWG-файл в AutoCAD версии 15 и старше как в однодокументном, так и
многодокументном режиме
|;
  (if (= 0 (getvar "SDI"))
;|-------------------------------------------------------------------
Для многодокументного режима (SDI=0) открываем новый документ в новом окне.
Сохранять открытые не нужно
-------------------------------------------------------------------;
    (vla-activate (vla-open (ru-obj-get-docs-collection) dwg_file))
    (progn
;|-------------------------------------------------------------------
Для однодокументного режима (SDI=1) нужно позаботиться о сохранении открытого
документа
-------------------------------------------------------------------;
  (if (not (equal 2 (logand 2 (getvar "QAFLAGS"))))
      (setvar "QAFLAGS" (+ (getvar "QAFLAGS") 2)))
 (if (not (equal 4 (logand 4 (getvar "QAFLAGS"))))
     (setvar "QAFLAGS" (+ (getvar "QAFLAGS") 4)))
;|-----------------------------------------------------
Мы использовали биты переменной QAFLAGS:
 бит 0=1 снимает выделение "ручек" (аналогично ESC)
 бит 1=2 отменяет паузы при выводе текста на экран
 бит 2=4 запрещает вывод диалогового окна функции alert
-----------------------------------------------------|;
    (command "_.OPEN")
;;; Проверяем, не изменялся ли текущий рисунок
;;; и, если изменялся - передаем опцию "Yes", дескать, сохраняй
  (if (not (equal 0 (getvar "DBMOD"))) (command "_y"))
    (command dwg_file)
    ;;; и теперь, пока работает команда OPEN,
    ;;; на все возможные запросы даем пустой ответ
    (while (wcmatch (getvar "CMDNAMES") "*OPEN*")
      (T (command ""))
    );_ end of while
    (setvar "QAFLAGS" 0)
  );_ end of progn
 );_ end of if
);_ end of defun
```

Выбор и вставка типового проектного решения

Вставить файл ТПР несложно, но нужно показать его свойства (это не заранее известный блок из меню). Реализуется программой, приведенной в листинге 33.17.

Листинг 33.17. Файл ru_doc_project_insert.lsp

```
(defun START (/ file_name projects_dir)
  (ru-app-begin)
  (if (ru-user-action-enabled "GET_PROJ" T)
    (progn
      (setq projects_dir (ru-dirs-archive-projects-dir))
      (princ "\nЖдите, загружаю навигатор...")
      (if
        (setq file_name
               (ru-dlg-file-select-or-new-dwg
                 "Выбор типового проектного решения"
                 (ru-user-read-last-param "LastProjectsDir" projects_dir)
                 projects_dir "" nil
               );_ end of ru-dlg-file-select-or-new-dwg
       );_ end of setq
        (progn
         (ru-block-insert-project (car file_name))
         (ru-user-write-last-param "LastProjectsDir"
                          (ru-file-path (car file_name)))
        )
     );_ end of if
   );_ end of progn
 );_ end of if
  (ru-app-end)
  (princ)
);_ end of defun
(START)
```

Непосредственно перед вставкой выбранного файла (листинг 33.18) мы получаем из него список свойств, показываем в диалоговом окне (рис. 33.9, листинг 33.19), выясняем единицы ТПР, правильно масштабируем и позволяем выполнить множественную вставку блока.

Листинг 33.18. Файл ru-block-insert-project.lsp

```
(defun ru-block-insert-project (dwg_name / note_base_point dwg_props
scale unit)
  (if (findfile dwg_name)
    (progn
      (setq dwg_props (ru-dwgprops-obj-get-from-file dwg_name))
      (ru-dwgprops-file-show dwg_name dwg_props)
      (if (setq note_base_point
                (cdr (assoc 302 dwg_props))
        );_ end of setq
```

```
        (setq note_base_point (ru-string-right-part note_base_point "="))
    );_ end of if
     (if (setq unit
                (cdr (assoc 301 dwg_props))
         );_ end of setq
       (setq unit (ru-string-right-part unit "="))
       (if (ru-yes
             (strcat
               "Для выбранного типового проекта '"
               "'\nне указано название единиц рисунка!"
               "\nПринять единицу рисунка '"
               (ru-unit-name)
               "'(как в текущем)"
             );_ end of strcat
          );_ end of ru-msg-yes
         (setq unit (ru-unit-name))
         (if (= (ru-unit-name) "м")
           (setq unit "мм")
           (setq unit "м")
         );_ end of if
       );_ end of if
    );_ end of if
     (if
       (= unit "м")
        (setq scale (ru-conv-meter-to-unit 1))
        (setq scale (ru-conv-millimeter-to-unit 1))
    );_ end of if
     (if note_base_point
       (princ
         (strcat "\nИзвестное описание точки вставки ТПР: '"
                 note_base_point
                 "'\n"
         );_ end of strcat
       );_ end of princ
    );_ end of if
     (ru-block-multi-insert-scaled-rotated-or-angleask
       dwg_name scale scale nil
    )
   );_ end of progn
    (princ (strcat "\nНе найден " dwg_name))
 );_ end of if
)
```

Листинг 33.19. Файл ru-dwgprops-file-show.lsp

```
(defun ru-dwgprops-file-show (caption dwg_props)
  (setq dwg_props (ru-dwgprops-test-list dwg_props))
  (ru-dlg-double-list caption "Переменная" "Значение"
    (mapcar 'ru-dwgprops-rus-name-by-code (mapcar 'car dwg_props))
    (mapcar '(lambda (x) (ru-string-right-part x "="))
```

```
            (mapcar 'cdr dwg_props)
    );_ end of mapcar
     T
  );_ end of ru-dlg-double-list
);_ end of defun
```

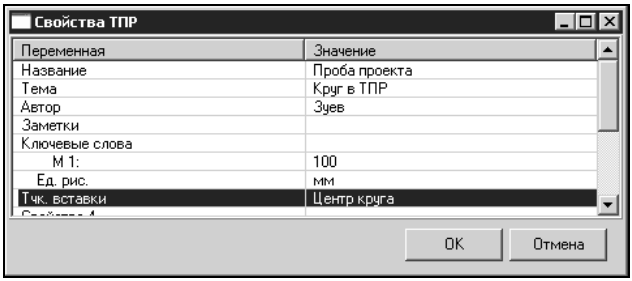


Рис. 33.9. Просмотр свойств типового проектного решения

Работа с DWF-файлами

Начиная с версии AutoCAD R13, фирма Autodesk внедрила замечательное решение для публикации чертежей — формат DWF (Design Web Format). На момент написания этой главы действует версия формата DWF6.

DWF — это формат файла для стандартизированного описания базированных на векторах, рисунков и иллюстраций. Поскольку DWF имеет обобщенный векторный формат, а не использует структуры данных создавшего его приложения (например, AutoCAD), то обеспечивается независимость от приложения. Совместимость обеспечивается посредством установки общего, расширяемого синтаксиса для обмена графическими данными между приложениями, генерирующими рисунки, и приложениями, читающими DWF-файлы.

В формате DWF данные могут храниться как в текстовом, так и в двоичном виде. Имеется метод сжатия данных, который минимизирует дублирование геометрической информации. Например, исходный файл формата DWG с планом территорий г. Кургана имеет размер 2,4 Мбайт, файл формата DWF с этого рисунка имеет размер 0,54 Мбайт. Очень удобно, что DWF включает в себя механизм для присоединения данных любого вида (с помощью накладывания связи или операции вложения). Поддерживается вложение URL-гиперсвязей в данные рисунка. Это позволяет использовать очень интересную возможность — навигацию по смежным DWF-файлам и даже по базам данных.

Например, в исходных DWG-файлах геоинформационной системы к объектам плана города мы, программным путем, привязывали расширенные данные вида "КОД_БД\КОД_ТАБЛИЦЫ\КОД_ЗАПИСИ". Информация об объекте хранится в базе данных, при работе с DWG-файлом специальная программа при указании на изображение объекта выводит в специальном диалоговом окне информацию из базы данных по этому объекту. При экспорте DWG-файла в DWF-формат в нем сохраняются все гиперссылки. Из базы данных экспортируются данные в формат HTML,

в одном из двух вариантов — один файл со всеми записями или много файлов с одной записью в каждой. В автоматически генерируемый DWF-файл для объектов привязываются автоматически генерируемые URL[1]. В результате при размещении DWF-файла на Web-странице возможен просмотр информации по объекту при щелчке по его изображению. Если информация в базе данных может изменяться часто и нерационально публиковать ее на статических страницах, то этот же DWF-файл при изменении обработчика события `OnNavigate` может обращаться за актуальной информацией непосредственно к базе данных, размещенной на удаленном сервере.

DWF-файлы быстро загружаются и просматриваются. Панорамирование и зумирование выполняются практически мгновенно.

DWF, в отличие от DWG, не предоставляет все данные файла рисунка широкой публике. Это означает, что рисунок в формате DWF можно просматривать (в лучшем случае, отпечатать), но нельзя редактировать. В DWF не попадает информация, находящаяся на отключенных слоях. Ваша интеллектуальная собственность надежно защищена.

Создание DWF

Вначале DWF создавали командой AutoCAD DWFOUT (ЭКСПОРТВ), которая имела несколько простых опций. Это было очень удобно для автоматического создания DWF.

По мере развития формата DWF и его возможностей фирма Autodesk перешла к изготовлению DWF-файлов путем вывода на виртуальный плоттер. Вывод на плоттер позволяет использовать множество тонких настроек и получать превосходные DWF-файлы[2]. Команда DWFOUT еще использовалась в AutoCAD 2002, но это была уже не "настоящая" команда, а оболочка-эмулятор, написанная на LISP[3] и фактически "вычисляющая" имя файла конфигурации плоттера; в AutoCAD 2004 от нее уже отказались (заменили на команду PUBLISH (ПУБЛ)). Недостаток вывода на плоттер заметен программистам, т. к. нельзя программно задать все требуемые параметры, можно только подготовить одну или несколько конфигураций плоттеров.

Для создания DWF-файлов могут быть использованы и отдельные инструменты, устанавливаемые независимо от системы AutoCAD. Особенно интересен появившийся в феврале 2004 года драйвер принтера для Windows **Autodesk DWF Writer**. Использование драйвера стандартного принтера позволяет создавать DWF-файлы из любого приложения. Недостаток мы пока видим в том, что в таком DWF не поддерживаются слои (даже если вывод производится из системы AutoCAD). Однако это позволяет создавать и хранить копии различных документов в едином формате "только для чтения", что очень удобно для систем документооборота.

В самой же системе AutoCAD имеется стандартная команда PUBLISHTOWEB (ОПУБЛИКОВАТЬ), позволяющая легко создавать HTML-страницы для публика-

---

[1] Uniform Resource Locator (URL, гиперссылка) — это строка специального вида, вставляемая в документ и позволяющая программе просмотра (такой, как Internet Explorer) при щелчке по URL открывать ресурс, на который ссылается URL.

[2] Отличный пример — %Acad2004%\Sample\Welding Fixture-1.dwf.

[3] %Acad2002%\Support\Legacy\DWFOut\dwfout.lsp.

ции чертежей. Программа рассчитана на неподготовленных пользователей, которые с помощью Мастера смогут и создать DWF-файлы, и разместить их на страницах. Генерируемые страницы имеют непритязательный дизайн, но технически написаны грамотно. Для изменения дизайна страниц можно просто откорректировать шаблоны таблиц стилей, страниц и скриптов, находящиеся в папке %Acad2004% \UserDataCache\Template\PTWTemplates.

Автоматически генерируемые страницы являются хорошим пособием для изучения технологии публикации чертежей.

Просмотр DWF

Просмотр DWF возможен несколькими программами. Самой удобной, на наш взгляд, является Autodesk Express Viewer, имеющая, кстати, русскоязычный интерфейс[1]. Больше возможностей имеет программа Volo View (поддержка стилей печати, пометок в документах, трехмерных видов и вращения по орбите), но это уже коммерческий продукт. Недостаток программы Autodesk Express Viewer нам видится в том, что ее авторы от версии к версии идут по традиционному пути все большего наполнения программы излишними возможностями, но этот недостаток компенсируется наличием хорошо документированного интерфейса прикладного программирования (API)[2], позволяющего создавать собственные программы просмотра.

Использование DWF

Мы используем DWF-файлы очень широко:

- для публикации в Интернете электронных карт;
- для публикации проектов;
- для предъявления заказчикам на предварительный просмотр выполненных проектов[3];
- в качестве "этикеток" к DWG-файлам.

Об DWF-этикетках

Прикиньте, как часто вы загружаете DWG-файлы в AutoCAD только для того, чтобы их посмотреть. Наша предыдущая система BestIA автоматически делает к каждому рисунку так называемую "этикетку" в виде DWF-файла. Файловый навигатор BestIA, с помощью которого открываются файлы, имеет средства для просмотра этикеток. Это позволяет избежать загрузок рисунков в AutoCAD.

Как создать DWF на LISP

В книжную версию системы ruCAD мы не включили функции для создания "DWF-этикеток" к рисункам — исключительно ради экономии места. Программно экспортировать чертеж в DWF-формат можно функцией, приведенной в листинге 33.20.

---

[1] **www.autodesk.ru/adsk/servlet/index?siteID=871736&id=3593406**.

[2] **www3.autodesk.com/adsk/files/3431901_Autodesk_Express_Viewer_API0.zip**.

[3] При заключении договоров на передачу проектных материалов в электронном виде мы передаем заказчику проект в виде DWF. Заказчик знакомится с проектом, оплачивает работы, после чего получает DWG-файлы.

Листинг 33.20. Функция `ru-dwf-plot`

```
(defun ru-dwf-plot (config_file plot_file /  plot orgqem)
  (setq plot       (vla-get-plot (ru-obj-get-active-document))
        orgqem (vlax-get-property plot "QuietErrorMode")
 );_ end of setq
    (vlax-put-property plot "QuietErrorMode" :vlax-true)
    (vlax-invoke plot "plottofile" plot_file config_file)
    (vlax-put-property plot "QuietErrorMode" orgqem)
);_ end of defun
```

Аргументами функции являются имена файла конфигурации и выходного DWF-файла. Все хитрости настройки находятся в файле конфигурации, создаваемом стандартными средствами системы AutoCAD. Подобную функцию можно встроить в любую программу.

DWF или PDF?

Довольно часто возникают вопросы о том, в каком формате сохранять документы для публикации или для передачи заказчику — в рассмотренном нами формате DWF или в формате PDF фирмы Adobe. Формат PDF в последние годы стал своеобразным стандартом для публикации документов. Фирма Adobe бесплатно распространяет программу просмотра PDF-файлов, продвигая тем самым коммерческие продукты для создания файлов такого формата. Документы в формат PDF обычно выводятся через драйвер принтера. В некоторых бесплатных продуктах имеется возможность экспорта в формат PDF.

Фирма Autodesk, конечно, заинтересована во внедрении своей технологии. Используя систему AutoCAD, мы легко, без дополнительных затрат, можем экспортировать данные в DWF, к тому же сохраняя важнейшие параметры исходного чертежа — слои, именованные виды, координаты объектов, компоновки листов.

Вот еще некоторые технические подробности (табл. 33.1) для файла willhome.dwg[1].

Таблица 33.1. *Сравнение технологий DWF и PDF*

Показатель	DWF	PDF
Размер файла, Кбайт	89	1624
Время создания, сек.	4	85
Размер дистрибутива программы просмотра, Мбайт	2,1	15,3/8,7
Пределы разрешения, точек на дюйм	15–60 000 000	72–4 000
Скорость рендеринга (файл sascatch32.dwg), сек.	0,1	7

[1] **usa.autodesk.com/adsk/servlet/index?siteID=123112&id=3523485**.

Проверить эти данные фирмы Autodesk в части формата PDF мы не можем, т. к. не собираемся специально покупать программу для создания PDF. Это, пожалуй, главное преимущество формата DWF для пользователей системы AutoCAD — "родной" формат, создаваемый "родной" программой. Один из бесплатных драйверов печати в PDF-формат для файла Welding Fixture-1.dwg создал PDF-файл очень хорошего качества, но размером 19,943 Мбайт, тогда как размер Welding Fixture-1.dwf такого же качества — 653 Кбайт.

Эти характеристики мы привели для того, чтобы в возникающих иногда спорах о достоинствах или недостатках технологий спорщики оперировали не понятиями "нравится", "стандартнее", "есть на каждом диске", а количественными и качественными показателями.

Резюме

Рассмотренные кратко технологии позволяют на базе нашей системы создать и систему документооборота. Ну, пусть не на нашей базе, а на базе какого-нибудь "дорогущего" решения, "проталкиваемого" системными интеграторами, но мы уже даем инструменты, способствующие наведению хотя бы элементарного порядка в электронном чертежном хозяйстве.

ГЛАВА 34

Интеграция САПР и ГИС

Эта глава появилась в книге, посвященной разработке САПР на базе системы AutoCAD, не случайно. Многие разделы проекта (генплан, транспорт, инженерные сети) разрабатываются на топографической основе. Естественно, что разработчикам этих разделов проекта, помимо средств для рисования своих изображений, нужно иметь и актуальную электронную топографическую подоснову. Чаще всего электронный план приходится или "доставать" (иногда незаконными методами), или сканировать имеющуюся топографическую основу и рисовать поверх полученного растра. При наличии дигитайзера возможна *оцифровка* топографических планов с помощью этого устройства (в прежние годы применялись и "шаманские" приемы "сколки" подосновы с помощью измерительного инструмента и общеизвестных заклинаний).

В то же время существует параллельный "мир ГИС", о существовании которого специалисты "мира САПР" часто не подозревают. А в этом параллельном мире, где широко используются компьютерные карты и планы, также не могут решить вопросы с получением достоверной информации. САПР и ГИС просто обречены на слияние этих двух технологий.

Что такое ГИС

Упрощенно[1] *геоинформационную систему* (ГИС) можно определить как комплекс электронных карт[2] и баз данных, в котором любые объекты, изображенные на карте, связаны с информацией, относящейся к этим объектам и описывающей их свойства. В простейшем случае каждому картографическому объекту ставится в соответствие строка таблицы (запись в базе данных) с атрибутивной информацией.

Город, как известно, начинается со стройки, а стройка — с проекта. Все, что в городе проектируется, когда-то (пусть с отступлениями от проектов) будет построено. Пользователям ГИС необходимо знать не только то, что в городе *есть*, но и то, что

---

[1] Определений термина ГИС существует примерно столько, сколько ученых занимается этой тематикой.

[2] Картами, в общих рассуждениях о ГИС, мы будем называть и собственно карты — изображения земной поверхности с учетом формы Земли, и планы, не учитывающие кривизны поверхности планеты.

будет (а зачастую и то, что *было*). Следовательно, данные проектов, по крайней мере в части генплана и коммуникаций, должны сразу попадать на планы ГИС в слои класса *будет*, по мере реализации генпланов переноситься (с фиксированием реального состояния) в слои класса *сейчас*.

Помимо очевидного эффекта от координации проектных решений (о, сколько же выявляется неувязок, при совмещении проектов на едином плане!), от интеграции САПР и ГИС появляются дополнительные преимущества, а именно: проектировщики получают возможность работать на готовой электронной топографической подоснове, а банк данных ГИС интенсивно пополняется новой информацией.

Возможности ГИС

ГИС позволяет отобразить любой участок местности (от карты мира до плана здания или помещения) в требуемом виде с необходимым набором видимых слоев. ГИС позволяет выбрать для отображения только интересующие в данный момент картографические слои, задать удобный порядок наложения этих слоев друг на друга, отобразить их на экране в удобном масштабе и желаемой картографической проекции, визуализировать на экране интересующий фрагмент карты, если вся карта не помещается на экране, вывести на карту пояснительные надписи или, наоборот, убрать их.

ГИС позволяет, указав объект на карте, получить ответ на вопрос "Что это?". При этом будет выведена описательная информация объекта из базы данных (такая, какая туда "заложена", в том числе, при необходимости, в виде фотографий).

ГИС позволяет, выбрав по каким-либо условиям объекты в базе данных, получить ответ на вопрос "Где это?". При этом будут указаны объекты или их местоположение на карте.

ГИС позволяет выбирать объекты по различным пространственным критериям (удаленность, нахождение в заданном контуре, круге и т. п.).

ГИС позволяет легко определять площади, расстояния, объемы сооружений, выполнять различные расчеты, например, последствий аварий на химически опасных объектах с отображением результатов на карте.

ГИС позволяет составлять различные тематические карты с отображением, например, районов области различными цветами и штриховками в зависимости от какого-то параметра (рис. 34.1).

ГИС различного уровня могут использоваться как для управления регионом, так и для производственных задач на уровне предприятия или даже отдельного здания. ГИС обладают способностью выявлять скрытые взаимосвязи и тенденции, которые трудно или невозможно заметить, используя привычные бумажные карты и таблицы.

Мы не будем рассматривать различные области применения ГИС — один только перечень реально используемых тематических карт может занять несколько десятков страниц. Коснемся только вопросов, близких к основной теме книги — системам автоматизированного проектирования — САПР.

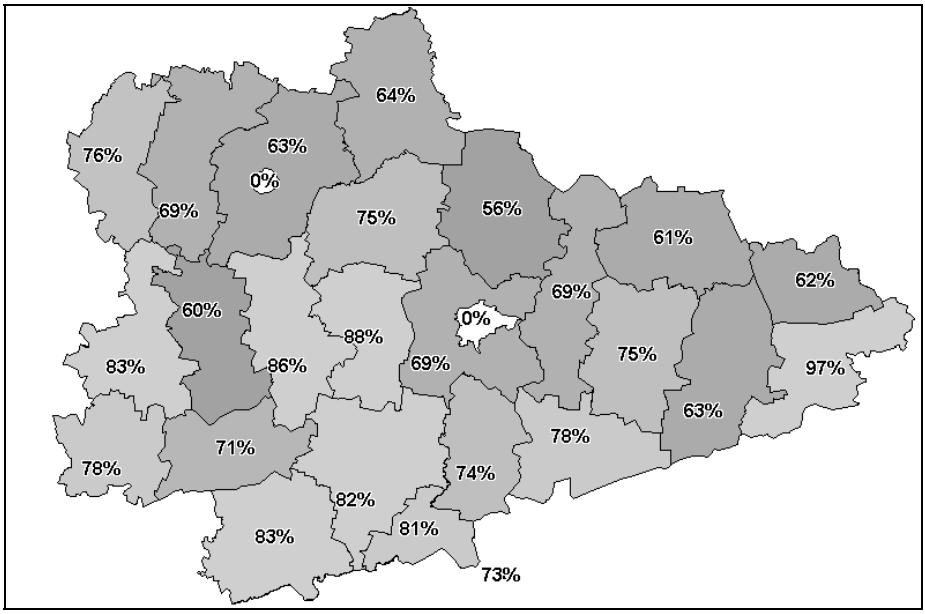


Рис. 34.1. Отображение на электронной карте хода уборки урожая

Составные части ГИС

Термин "геоинформационная система" (ГИС) обычно применяется в двух смыслах.

Во-первых, ГИС часто называют какое-то конкретное программное средство, например: "ГИС ИнГео". При всем нашем уважении к этому замечательному продукту, это все-таки не геоинформационная система, а только инструментальное средство для ее создания. В рекламе продуктов фирмы Autodesk применяются более корректные (если не обращать внимания на прилагательное "уникальный") наименования: "AutoCAD Map — уникальный инструмент для автоматизированного картографирования и ГИС", "Autodesk World — уникальный интегратор географических данных", "Autodesk Map Guide — уникальная интернет-технология для ГИС".

Во-вторых, ГИС называют некую информационно-справочную или информационно-аналитическую систему в какой-либо предметной области. Вот такая ГИС и нужна конечным пользователям — это может быть "ГИС администрации города Арбатов" или "ГИС ООО "Рога и Копыта"".

Истинные геоинформационные системы состоят из следующих компонентов:

- *аппаратные средства* (компьютеры, сканеры, дигитайзеры, плоттеры);
- *программные средства* — инструменты для создания ГИС, обработки данных и предоставления результатов конечным пользователям;
- *информация* — электронные карты и пространственно координированные данные, как правило, связанные с изображениями на картах;

- *пользователи* — специалисты, создающие и поддерживающие информационную систему в актуальном состоянии, и прочие пользователи, использующие информацию в собственных целях;
- *технологии* — методы создания карт и данных и дальнейшей работы с ними.

Аппаратные средства мы рассматривать не будем — любой современный персональный компьютер может работать с геоинформационной системой. Не будем рассматривать и программные средства для решения узкого круга специфичных задач (например, обработки аэрофотоснимков).

Все остальные составные части оказывают существенное влияние на конечный результат — ошибка в любой составляющей может привести к полной дискредитации информационной системы. Особое значение имеют технологии работы, т. к. от выбора того, *что* и *как* делать, зависит принятие решения о том, *чем* делать. Весьма распространенной ошибкой является приспосабливание конечных целей под возможности инструментального средства — делают, *что могут* и *как могут*, вместо того, *что нужно* и *как нужно*. Виновниками часто становятся программисты, которые якобы "могут все сломать" (для начальников это кажется признаком квалификации) и с энтузиазмом берутся за решение задач, в которых ничего не понимают.

Инструментальные средства ГИС

Сейчас имеется множество программных средств для создания геоинформационных систем. Особенно отрадно, что на рынке появились программные продукты российских разработчиков, не уступающие именитым зарубежным конкурентам. Но наша книга посвящена САПР, и рассматривать мы будем только продукты, позволяющие осуществить интеграцию САПР и ГИС. Конечно, существуют различные САПР-платформы, на базе которых возможны интегрированные решения, но все-таки лидером САПР в России остается система AutoCAD. От фирмы Autodesk долго ожидали и аналогичных решений в области ГИС, и они последовали. Появилась целая линейка продуктов для работы в ГИС — AutoCAD Map, Autodesk World, Autodesk Map Guide, Autodesk Land Desktop и прикладные приложения.

Однако один из главных лозунгов фирмы Autodesk "Сделаем работу с геоинформацией доступной всем!" все-таки вызывает сомнение.

Во-первых, смущают цены. Мы уже прикидывали, во что обойдется проектной организации оснащение рабочих мест полным набором продуктов на базе AutoCAD *(см. главу 2)*, и сказали: "Однако!" Но если на рабочих местах проектировщиков производится проектная продукция, продаваемая заказчикам, и САПР-продукты, в конце концов, себя окупят, то с ГИС положение иное. ГИС, по определению, является информационной системой, предназначенной для предоставления информации по возможности широкому кругу организаций и граждан. На одно рабочее место "создателя" ГИС приходится несколько десятков, а то и сотен рабочих мест пользователей, большинству из которых информация должна предоставляться бесплатно. Если укомплектовать пару рабочих мест продуктами фирмы Autodesk в головной организации, создающей муниципальную ГИС, еще сгоряча возможно, то как заставить сделать это десятки организаций, которые тоже должны принимать участие в создании такой ГИС, иначе грош ей цена? Значительные затраты требуются на создание

и поддержание информации в актуальном состоянии, а продавать практически ничего (по крайней мере в первые годы создания ГИС) нельзя.

Во-вторых, продукты фирмы Autodesk требуют значительной доработки. Хотя иногда можно встретить утверждения, что "это (AutoCAD Map) самая дешевая полнофункциональная ГИС", но мы то с вами знаем, что и базовая система AutoCAD, и "десктопы" — не конечные продукты "под ключ", а скорее "шасси", которые нужно доукомплектовывать предметными приложениями. Не являются исключениями и ГИС-продукты фирмы Autodesk. У них имеется множество функциональных возможностей, которые никогда не понадобятся пользователю, и нет средств для решения первоочередных задач.

> Когда-то мы с нетерпением ожидали появления анонсированного AutoCAD Map, наивно полагая, что в продукте с таким названием появятся специализированные средства для рисования планов и карт. Увы! Появился дежурный набор, уже реализованный в известных продуктах — оцифровка, векторизация, коррекция топологии, семьсот проекций, тематическая картография плюс весьма неудобная технология связки графической и семантической информации, и ничего из ожидаемых инструментов. *(Сергей Зуев)*

Даже в статьях, посвященных внедрению таких продуктов, подтверждается — "мы рассматриваем AutoCAD Map 2000 не как конечный продукт, а как средство разработки, для решения специальных задач". Возможность разработки собственных расширений одна из причин успеха системы AutoCAD, но эти работы также требуют времени и средств.

В-третьих, система AutoCAD и продукты на ее основе сложны в освоении. Не зря издается столько книг, посвященных различным аспектам работы с AutoCAD. ГИС-продукты фирмы Autodesk, в отличие от конкурирующих систем, не русифицированы, не имеют документации на русском языке. Осваивать ГИС-технологии в таких условиях сложно. Конечно, "продвинутые" пользователи и разработчики ГИС преодолевают эти трудности, но попробуйте обучить работе с такими программами "тетку", особенно если она занимает должность мэра города или губернатора — а именно с облегчения работы руководителей такого уровня нужно начинать, чтобы обеспечить финансовую и моральную поддержку создаваемой системе.

Альтернативные инструментальные средства ГИС в рассмотренных отношениях выглядят лучше — и по цене, и по "готовности к употреблению", и по простоте освоения. Более того, нам кажется, что фирма Autodesk продвигает свои продукты в уже занятую нишу геоинформатики, использующей мелкомасштабные карты, недостаточно уделяя внимание той, где они могли бы быть вообще вне конкуренции — интегрированным с САПР информационным системам, базирующимся на точных крупномасштабных планах.

При использовании продуктов на базе системы AutoCAD в мелкомасштабных ГИС проявляются и его органические недостатки — невозможность управления порядком отображения слоев, отсутствие встроенной генерализации изображений, недостаточная цветовая палитра (сдвиги в лучшую сторону наметились только в AutoCAD 2004).

Однако известны и преимущества системы AutoCAD:

- ❒ огромное количество обученных пользователей;
- ❒ близкий к идеалу набор средств для рисования и редактирования;
- ❒ удобные инструменты для управления изображением;

- мощная система разработки приложений, которую мы подробно рассматриваем в книге;
- большое количество прикладных программ и библиотек;
- множество наименований технических изданий, рассчитанных на пользователей разных категорий.

Для того чтобы создать эффективную информационную систему, нужно использовать продукты по назначению там, где меньшее значение имеют их недостатки, но лучше могут проявиться достоинства. Это банальная мысль, но мы постоянно видим, как люди, берущиеся "сделать ГИС", делают это, не изучив вопрос всесторонне. Выбор инструмента для ГИС сложен, неоднозначен и ответственен. Именно поэтому мы отдаем столько места "пустым разговорам" на эту тему.

Область применения ГИС на базе AutoCAD

Руководствуясь собственным 10-летним опытом (и достижениями, и ошибками) разработки ГИС на базе системы AutoCAD, мы можем очертить круг рационального применения этой платформы.

Во-первых, мы рекомендуем ограничиться крупномасштабными планами — не мельче масштаба 1:2000, а еще лучше — 1:500. Именно на плановой основе таких масштабов ведется рабочее проектирование, для которого необходима самая точная подоснова. Масштаб электронной подосновы и масштаб рабочих чертежей должны быть одинаковыми. При этом, помимо крупномасштабных планов, созданных на основании топографических планшетов, должен быть относительно небольшой набор слоев с общей ситуацией по территории. Это могут быть и не очень точные схемы, но лучше, если они основываются на достаточно точных источниках, например на данных земельного кадастра.

Во-вторых, в дополнение к обязательному первому условию должна быть явная и осознанная заинтересованность руководителей города или предприятия в ведении именно точного плана города или предприятия (вплоть до отражения планировок зданий), пригодного для рабочего проектирования.

В-третьих, на обслуживаемой ГИС территории должно вестись достаточно активное проектирование и строительство, иначе в точных планах просто не будет потребности. Энтузиазм руководителей по финансированию работ быстро иссякнет, если не будет подкрепляться давлением и материальной помощью со стороны заказчиков.

Все эти условия могут соблюдаться в муниципальных ГИС и в информационных системах промышленных предприятий, причем для предприятий внедрение таких технологий реальнее из-за широкого применения AutoCAD в разных службах и отсутствия множества анархистски настроенных самостоятельных руководителей.

Однако, даже если в городе создается базовая электронная топографическая основа с использованием системы AutoCAD, это не означает, что все организации должны использовать продукты на этой платформе — отраслевые ГИС могут применять иные технологии. Как сделать данные, подготовленные в AutoCAD, доступными для его "непользователей", мы еще обсудим.

Конечно, с использованием линейки ГИС-продуктов фирмы Autodesk можно разрабатывать разнообразные ГИС (кадастровые, статистические, экологические, геогра-

фические, социально-экономические и пр.), но в большинстве случаев это не будет оптимальным решением.

Об электронных картах

Электронные карты являются обязательным элементом ГИС, но далеко не каждое изображение местности на экране компьютера обладает бóльшим, чем бумажная карта, информационным наполнением. Если это отсканированная или даже векторизированная модель бумажного топографического планшета, то на ней мы в лучшем случае увидим только то, что было на оригинале.

Информация об объектах на обычном плане представлена в виде надписей и условных знаках. По условному знаку можно, например, определить, что нарисованный "кружок с точкой" (рис. 34.2) это разведочная скважина, да еще узнать отметку устья и грунтовых вод.

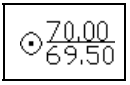
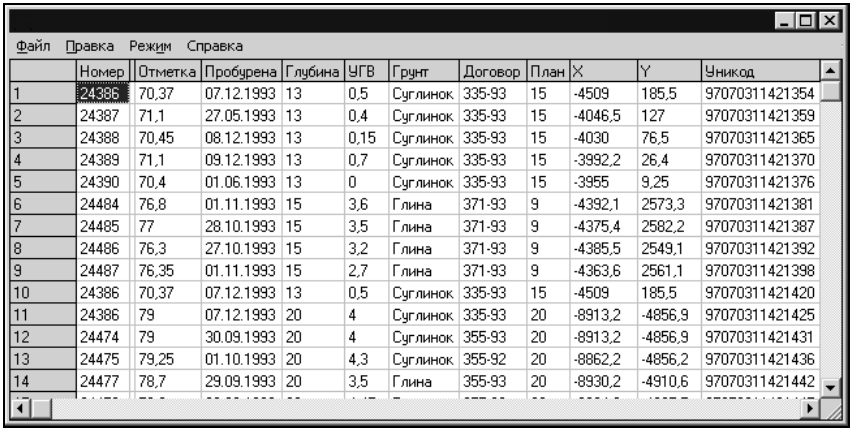


Рис. 34.2. Изображение скважины на плане

Если же к изображениям скважин привязана описательная информация ("семантика", на ГИС-жаргоне), то можно узнать об объекте очень много (рис. 34.3).


Файл Правка Режим Справка

	Номер	Отметка	Пробурена	Глубина	УГВ	Грунт	Договор	План	X	Y	Уникод
1	24386	70,37	07.12.1993	13	0,5	Суглинок	335-93	15	-4509	185,5	97070311421354
2	24387	71,1	27.05.1993	13	0,4	Суглинок	335-93	15	-4046,5	127	97070311421359
3	24388	70,45	08.12.1993	13	0,15	Суглинок	335-93	15	-4030	76,5	97070311421365
4	24389	71,1	09.12.1993	13	0,7	Суглинок	335-93	15	-3992,2	26,4	97070311421370
5	24390	70,4	01.06.1993	13	0	Суглинок	335-93	15	-3955	9,25	97070311421376
6	24484	76,8	01.11.1993	15	3,6	Глина	371-93	9	-4392,1	2573,3	97070311421381
7	24485	77	28.10.1993	15	3,5	Глина	371-93	9	-4375,4	2582,2	97070311421387
8	24486	76,3	27.10.1993	15	3,2	Глина	371-93	9	-4385,5	2549,1	97070311421392
9	24487	76,35	01.11.1993	15	2,7	Глина	371-93	9	-4363,6	2561,1	97070311421398
10	24386	70,37	07.12.1993	13	0,5	Суглинок	335-93	15	-4509	185,5	97070311421420
11	24386	79	07.12.1993	20	4	Суглинок	335-93	20	-8913,2	-4856,9	97070311421425
12	24474	79	30.09.1993	20	4	Суглинок	355-93	20	-8913,2	-4856,9	97070311421431
13	24475	79,25	01.10.1993	20	4,3	Суглинок	355-92	20	-8862,2	-4856,2	97070311421436
14	24477	78,7	29.09.1993	20	3,5	Глина	355-93	20	-8930,2	-4910,6	97070311421442

Рис. 34.3. Информация о скважинах в базе данных

Но электронная карта в ГИС способна на большее. Мы можем установить такой режим отображения карты, при котором скважины будут изображаться в виде пятен, имеющих цвет, соответствующий глубине залегания грунтовых вод (например, чем ближе к поверхности, тем интенсивнее оттенок синего цвета). Любой инженер-строитель знает, сколько времени нужно затратить на то, чтобы узнать, выполнялись ли на площадке инженерно-геологические изыскания и разыскать отчеты об этих работах. А с помощью ГИС-технологий мы сразу видим и общую информацию об

изученности территории, и, щелкнув по знаку скважины, получаем детальную информацию. Но у нас не простая карта, а тематическая, и инвестор, не вникая в технические детали, может заранее оценить предлагаемую площадку для строительства. Один взгляд на такую карту может спасти от значительных бесполезных расходов.

Аналитическая карта, показанная на рис. 34.4, сформирована с помощью программы Mapinfo, база данных скважин создана в системе BestIA, являющейся приложением к системе AutoCAD.

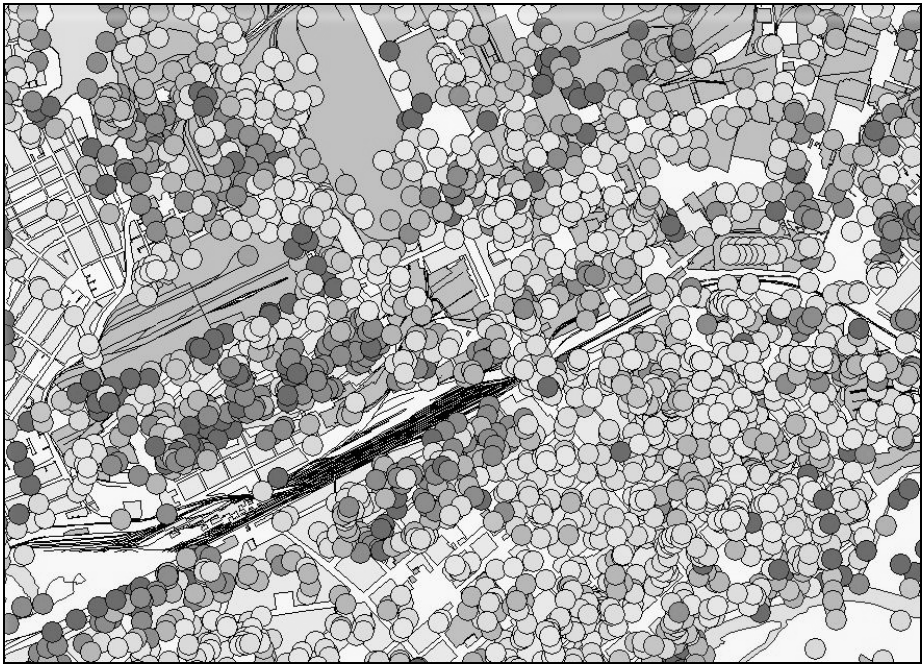


Рис. 34.4. Тематическая карта залегания грунтовых вод

Семантическая информация, привязанная к объектам карты, не образуется сама по себе в процессе векторизации. Помимо решения технических вопросов, как и какую "семантику" привязать, надо решить, где, а главное *как* эту информацию добыть[1].

Если на топографическом планшете М1:500 показаны каждый столб, дерево, сарай, забор или кабель, то, конечно, полезно было бы иметь характеристики этих объектов, но где же их взять? Хорошо еще, если можно раздобыть минимальную информацию об основных заданиях (адрес, этажность, владелец и т. п.), т. к. не вся она отображается на планах. Поэтому задачи нужно ставить реальные, распределять наполнение карты информацией во времени и между исполнителями.

Например, в органе архитектуры может поддерживаться топографическая информация, предназначенная для использования любыми организациями города. На топо-

[1] Приемы и методы добывания информации мы выносим за рамки книги.

графических планах показаны, в частности, тепловые сети (условным обозначением, в одну линию и не всегда с указанием диаметров труб). Подробная же информация о тепловых сетях с указанием всех их детальных характеристик должна поддерживаться владельцем сетей с передачей той части информации, которая может или должна быть доступна "смежникам" в информационную систему города.

Откуда берутся электронные карты?

Судя по тому, как мало внимания уделяется созданию электронных карт в зарубежных программных продуктах, все они уже давно созданы. В наших условиях создавать электронные карты приходится, в основном, самостоятельно, хотя имеется множество фирм, предлагающих такие услуги. К подобным предложениям со стороны иногородних фирм мы рекомендуем относиться очень осторожно. Самым лучшим вариантом является приобретение электронной карты в организации, являющейся официальным держателем топографических данных территории.

Модель карты или модель местности?

В мире ГИС, как и в САПР, постоянно циркулируют идеи об геоинформационных системах, основанных на трехмерной модели местности. Безусловно, это очень необходимо, но смысл в такой модели есть только тогда, когда она точная. Но как точно нанести, например, подземные коммуникации, если в исходных материалах в лучшем случае имеются отметки лотков канализационных колодцев?

В то же время в отдельных случаях, когда производится сквозное трехмерное проектирование нового объекта, проектные разработки могут послужить основой для ГИС.

У вас хорошая крыша?

Создание и ведение геоинформационных систем, создание тематических карт (в том числе в цифровой форме) относится к геодезической и картографической деятельности, которая в соответствии с федеральным законом "О геодезии и картографии" подлежит лицензированию. А у вас есть лицензия на такую деятельность? Если нет, то ее надо получать "в установленном порядке" в территориальном органе Госгеонадзора.

Где брать исходные материалы

Прежде чем бросаться "заносить в компьютер" карту или план, подумайте, стоит ли это делать именно с этим материалом.

Во-первых, проверьте легитимность карты. Откуда она появилась на вашем столе — куплена в магазине, "достали", "сами нарисовали"? В любом из этих случаев вы вступаете в конфликт с законом — нарушая авторские права создателя карты или занимаясь незаконной деятельностью. Даже если исходный материал получен от владельца, убедитесь, что имеется его письменное согласие на создание вторичных материалов.

Во-вторых, проверьте качество карты. Если она "отрэмлена" или даже "отксерена", то наверняка имеются значительные искажения, которые сделают электронную карту неточной.

Постарайтесь использовать оригинальные, а не вторичные материалы. Оригинальными являются топографические планшеты (на картоне, металле, лавсановой пленке), а вторичными — различные "выкопировки" и отчеты о топографических изысканиях. Вопрос с доступом к исходным материалам очень сложен, но его нужно решать.

Никогда не используйте в качестве подосновы имеющиеся в эксплуатационных организациях схематичные планы, хотя иногда они выглядят "как настоящие".

Привязка к системе координат

Любой топографический план должен быть привязан к местной системе координат! Если вы работаете с настоящим топографическим планшетом, то координаты углов планшета указаны в зарамочном оформлении. Если же вам вынужденно приходится работать с выкопировкой из плана, то на ней обязательно должны быть "крестики" координатной сетки, и хотя бы у одного из них должно быть значение координат. Не забывайте, что в геодезии ось, направленная на север, обычно[1] *называется* X, а не Y. Необходимо также знать, где физически (с привязкой к объектам местности) расположена точка начала координат. А как привязано начало местной системы координат к географическим вам лучше не знать.

Совет

В AutoCAD в качестве местной системы координат используйте мировую систему координат, и все объекты местности рисуйте только при установленной МСК.

План, не привязанный к местной системе координат, не пригоден для работы. Особенно ужасно, если план привязан неправильно — это хуже отсутствия координат.

Пример

В городе N служба, занимавшаяся нормированием выбросов в атмосферу, запросила в Горархитектуре обзорный план города, *не указав*, для каких целей он будет использоваться. Горархитектура, *не уточнив цели*, выдала старинную неточную специально искаженную схему. А экологи задумали использовать этот план для задания предприятиям точек начала местных систем координат, используемых при определении координат источников выбросов в проектах нормативов предельно допустимых выбросов в атмосферу (ПДВ). При этом, не долго думая, нарисовали в левом нижнем углу схемы точку начала координат. Начало координат местных систем задавалось предприятиям после измерения портновским измерительным инструментом от этого условного начала до не менее условного угла площадки предприятия. Поразительная техническая безграмотность экологов, не знающих о городской системе координат ("география — наука для ямщиков"), привела к тому, что сотни тысяч координат источников выбросов на десятках предприятий были установлены неправильно. Впоследствии эти "координаты" были добросовестно внесены в базу данных, и только при попытке показать источники на электронной карте "экологической ГИС" выяснилась истинная картина. Разумеется, в итоге пострадали предприятия, которых заставили переделывать все проекты ПДВ, запрашивая правильные привязки местных систем к городской системе координат.

---

[1] В некоторых городах осью X называют ось, направленную на восток. Незнание этой детали может привести к ужасным ошибкам, поэтому всегда уточняйте название осей на незнакомой территории.

Процесс создания электронной карты

Если вы все-таки решаетесь на создание электронной карты, изучите различные технологические варианты.

Замечание

Все последующие советы мы даем из предположения, что вы работаете в AutoCAD и используете описываемую в книге систему ruCAD.

Не следует сразу бросаться на векторизацию топографических планшетов — лучше создайте схематический обзорный план территории. Для таких целей подойдет даже туристическая карта, выполненная в масштабе (которому особо доверять не следует).

Выясните систему разграфки планшетов для вашей территории и начертите теоретическую сетку размещения планшетов. Совместите, по возможности точно, схему города с сеткой планшетов.

Совет

Используйте классификатор слоев, рассмотренный в *главе 18*, для распределения объектов по стандартным слоям. При необходимости пополняйте классификатор.

В дальнейшем схему города можно постоянно уточнять, повышая ее точность. Желательно использовать любые источники, в которых координаты встречаются в аналитическом виде:

- материалы земельного кадастра (координаты зон, массивов, кварталов и участков);
- каталоги координат красных линий;
- каталог координат границы городской черты;
- акты выноса объектов в натуру;
- каталоги разведочных скважин, пунктов полигонометрии;
- геодезические ходы, опорные точки, скважины;
- координаты источников выбросов в ведомостях инвентаризации;
- проекты детальной планировки и другие проектные материалы.

Таких источников информации имеется очень много, но беда в том, что надо знать, где они есть и как их взять. Здесь решающее значение имеет знание разработчиками ГИС всех тонкостей взаимоотношений между городскими организациями и личные связи (например, среди опытных топографов и генпланистов, многие из которых имеют личные архивы). Совокупность всех объектов, полученных из таких источников, дает прекрасную опорную сеть. Разумеется, просто так эти материалы вам никто не даст, поэтому надо заручиться поддержкой "самого главного" и, кроме того, создавать собственную "службу внешней разведки".

Создав более или менее узнаваемую схему города, ее нужно сразу использовать для демонстрации результатов.

Пример

Сформированную примерно так схему мы предоставили службе по чрезвычайным ситуациям, где она использовалась для учета потенциально опасных объектов и для отобра-

жения результатов расчетов прогнозируемых аварий. Это дало и дополнительный источник финансирования, и наглядно показало полезность ГИС.

Сразу же создавайте базы данных по соответствующей тематике и начинайте наполнять любой доступной семантической информацией.

После того как ГИС начнет работать, можно приступать к самой объемной работе по векторизации планшетов.

Идеальным является вариант, при котором электронные топографические планшеты создаются и поддерживаются в актуальном состоянии единой городской геодезической службой. К сожалению, наличие таких служб (в отличие от наличия обязательных городских сумасшедших) является исключением, а не правилом.

Векторизация планшетов

При векторизации планшетов обычно выполняются следующие операции:

- *сканирование* — получение фотографического изображения планшетов;
- *сводка* — размещение углов растров планшетов так, чтобы они совпадали с теоретическими координатами углов планшетов;
- *векторизация* — ручная, автоматическая или полуавтоматическая обрисовка объектов поверх растровых изображений планшетов;
- *формирование сводного плана из отдельных планшетов*;
- *топографический мониторинг* — внесение текущих изменений.

Каждая из этих операций имеет массу важнейших деталей, а трудоемкость во многом зависит от применяемых технологий. Например, можно потратить массу времени на мифическую автоматическую или реальную полуавтоматическую векторизацию путем программной обработки растра. А можно, располагая программными средствами, создание которых рассмотрено в нашей книге, очень быстро нарисовать объекты местности поверх растра.

Примеры

При оцифровке изображения откоса нет необходимости координировать каждый штрих, достаточно "сколоть" определяющие точки верхней и нижней подошвы и откос будет нарисован в полном соответствии со стандартом. Для отрисовки трассы канализации указывают только центры колодцев, а все остальное (сам колодец, примыкающие трассы с обозначениями) рисуется автоматически.

Подробнее остановимся на формировании сводного плана из отдельных электронных планшетов.

Совет

В системе AutoCAD каждый электронный планшет должен быть отдельным DWG-файлом со специальным именем, которое можно вычислить по координатам точки на местности. Базовой точкой вставки для всех планшетов должна быть точка начала МСК. Рамка планшета должна охватывать квадрат с соответствующими координатами. В электронном планшете должен быть полный набор соответствующих классифицированных слоев, а во всех планшетах одинаковые объекты должны размещаться на одноименных слоях.

Формирование сводного плана

Существует два способа создания сводного плана:

- вставкой "встык", в виде кафельной плитки отдельных планшетов, без объединения;
- объединение изображений на отдельных планшетах в единое поле.

Некоторые специалисты, зациклившись на деталях, пишут о технологиях векторизации, считая само собой разумеющимся, что планшеты могут получаться только векторизацией растров, а в результате *обязательно* должно быть получено *единое векторное поле*.

Например, в программном продукте ИнГЕО на очень высоком техническом уровне реализована технология СРВ (Сшивка растров с последующей приоритетной векторизацией объектов) — выборочная обрисовка контуров объектов, расположенных на разных планшетах. Мы можем засвидетельствовать — ИнГЕО, работает с этой технологией очень хорошо и действительно является примером "революционных технологических достижений в обработке информации персональными компьютерами[1]" и нравится нам больше, чем многие знаменитые программные продукты.

Однако в идеологию СРВ заложено ложное предположение, о том, что можно и нужно "*раз и навсегда* сшить получившиеся векторные изображения планшетов". Если мелкомасштабные планшеты и даже планшеты М1:2000 действительно практически не изменяются, то "пятисотка" корректируется постоянно, причем корректировка производится именно попланшетно. Сшить все планшеты в один план можно, но даже такая шустрая программа, как ИнГЕО, не справится с обработкой единого плана большого города. А если и справится (делает она это очень ловко благодаря тому, что вообще не хранит данные в графических форматах — все в БД), то практически невозможно отслеживать и вносить изменения, постоянно вносимые в оригиналы, т. к. при малейшем изменении любого объекта меняется вся база данных.

При хранении каждого планшета в отдельном файле просто не нужно и пытаться сшивать их ни в растровом, ни в векторном виде. Условно-постоянные объекты (например, контуры зданий), требующие привязки атрибутивной информации, должны просто располагаться на отдельных слоях, а еще лучше, если координаты их контуров хранятся в базе данных и доступны для использования и отображения в любых системах, а не только в AutoCAD или ИнГЕО.

Проектировщику, для того чтобы начертить план коммуникаций отдельного здания, не потребуется загружать план всего города, а достаточно путем нескольких щелчков мышью подгрузить только требуемые планшеты.

А вот собрать единый топографический план среднего завода вполне реально, но все изменения обстановки должны фиксироваться на этом плане.

Связь изображений с базами данных

Вторым, после электронных карт, важнейшим компонентом ГИС являются базы данных, в которых хранится семантическая информация об изображенных на карте

[1] Цитата из технической документации на ИнГЕО.

объектах. Именно связки изображений на экране с записями в базе данных делают обычную графическую систему информационной.

В различных программных инструментах для ГИС эти связки организуются собственными методами. Разберем, как это можно сделать в AutoCAD.

Самым простым способом, доступным LISP-программистам, является использование расширенных данных примитивов. В своих системах мы программным путем привязывали к примитивам строку расширенных данных специального формата, в которой записывались через символы-разделители имена базы данных, таблицы и уникальный код записи, например:

```
"GIS\TS\MKRN12;ТЕПЛОПУНКТЫ;99111811500323".
```

Возможны и более изощренные схемы, но в любом случае внутри DWG-файла хранятся связанные с примитивами указатели на записи в базе данных.

Важнейшим является вопрос о том, что же хранится внутри самой базы данных. Возможны две принципиально отличающихся технологии:

- хранение пространственной информации в графическом файле;
- хранение пространственной информации в базе данных.

Вариант пространственной информации в графическом файле

При использовании первого варианта в базе данных может быть записана различная информация. Кроме того, любая таблица с произвольной структурой может быть привязана к любому примитиву AutoCAD. Примером является база данных %ACAD%\Sample\db_samples.mdb с набором включенных в нее таблиц. Единственным подозрительным полем является поле `Entity_Handle` в таблице `Room`. В данном случае для связки используется номер метки примитива. Такой методический подход удобен тем, что не накладывается никаких ограничений на структуру базы данных. Можно использовать любые существующие базы данных (обычно в каждой есть какое-то уникальное поле) для привязки к изображениям. Но эта технология обладает большим недостатком — невозможно определить пространственное положение объектов, характеристики которых хранятся в базе данных при недоступности DWG-файла. Восстановить связь примитив-запись можно только с использованием технологий фирмы Autodesk, т. е. используя весьма дорогие программные продукты. Фактически мы сталкиваемся с насильственной привязкой пользователей к продукции фирмы[1]. Конечно, с точки зрения фирмы это несущественно, но возникают и технические проблемы.

Во-первых, редактирование или даже только просмотр DWG-файлов в истинно многопользовательском режиме невозможно или чрезвычайно затруднено.

Во-вторых, в ГИС обязательно необходим поиск по пространственным критериям, а это опять привязка к выбору и фильтрации объектов внутри системы AutoCAD.

---

[1] Большинство других фирм также применяют подобные приемы.

Вариант пространственной информации в базе данных

При работе по второму варианту вся информация, требуемая для восстановления геометрии объектов, хранится в базе данных. Взгляните на рис. 34.5 — на нем показан редактор пространственной базы данных. Мы видим не только основные семантические характеристики объекта, но и его контур со списком координат вершин, которые можно редактировать. Первоначально контур здания был нарисован в системе AutoCAD полилинией, затем список координат был отправлен в базу данных, в базе данных была создана запись с уникальным кодом, заполнены описательные поля, а после завершения редактирования к полилинии были привязаны расширенные данные с указателем на запись в базе данных. Если при редактировании в БД будет изменен список координат, то полилиния будет перерисована по новому контуру.

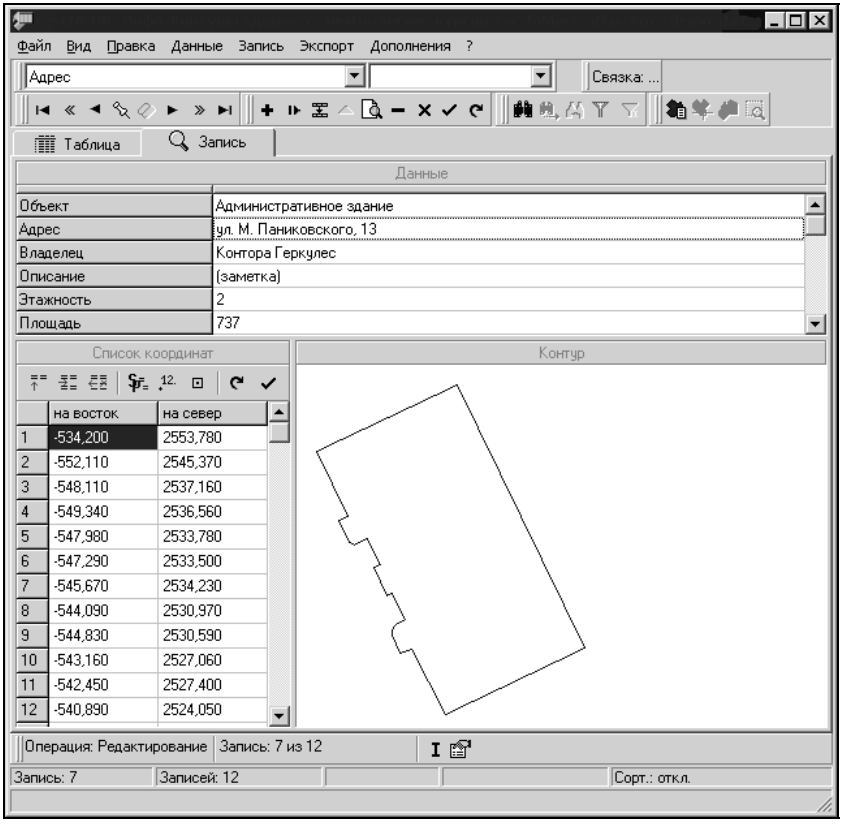


Рис. 34.5. Пространственно координированная база данных

Создав с помощью системы AutoCAD необходимый набор объектов в базе данных (например, внеся все здания города), мы можем вообще забыть о ее существовании!

Отобразить контуры по известным координатам мы можем в любой графической системе. Заметим, что и дополнять базу данных мы также можем из любой программы, которая "знает", как передавать в нее координаты.

Конечно, пользователи, использующие систему AutoCAD (с нашими или собственными небольшими "ГИС-довесками"), по-прежнему могут использовать ее мощь и несомненные преимущества, но информация теперь будет доступна и пользователям иных систем. Технология работы с базами данных в многопользовательском режиме отработана и надежна. Имея в базе данных списки координат объектов, можно организовать и поиск по пространственным критериям ("найти информацию по кабелям, проложенным ближе 5 м от такого-то здания" и т. п.).

Недостатком хранения пространственной информации в базе данных является наличие специальных полей для хранения координат, но этот недостаток компенсируется гибкостью и дополнительными возможностями.

Еще один пример использования такой базы данных показан на рис. 34.6 (к сожалению, черно-белое изображение не способствует восприятию карты).

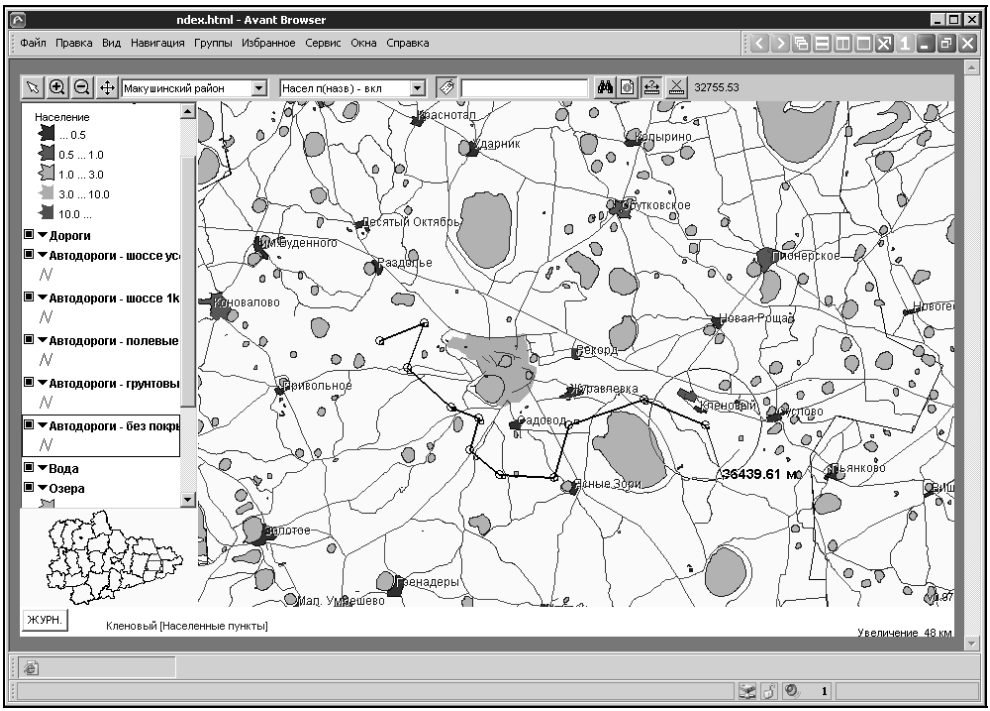


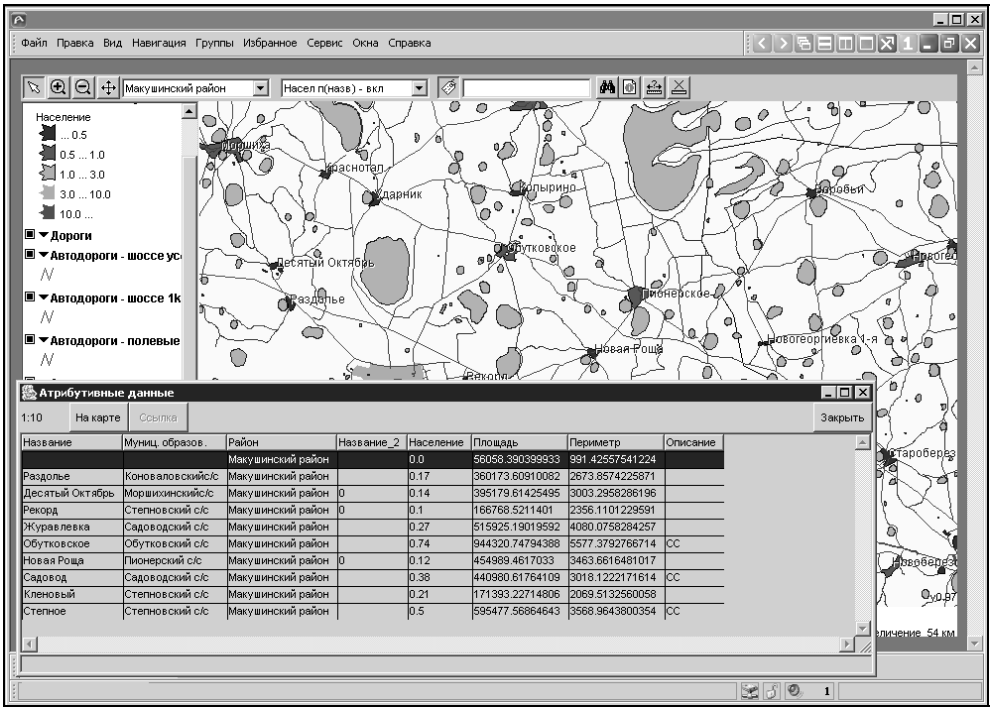
Рис. 34.6. Векторная электронная карта, просматриваемая в Интернете

На рисунке показана электронная карта области, просматриваемая через браузер (адрес скрыт, т. к. доступ к карте пока ограничен). Просмотр карты осуществляется в режимах, привычных для пользователей САПР и ГИС (зумирование, панорамирование). Возможно управление видимостью слоев, причем вид слоев управляется

с помощью XML, на котором и написана просматриваемая страница. Например, цвет полигонов населенных пунктов зависит от численности населения, извлекаемой из базы данных. Возможно управление видимостью подписей, например, названий населенных пунктов. Внешний вид карты каждый пользователь может настроить по своему желанию. В левом нижнем углу показано окно общего вида, хорошо знакомое пользователям AutoCAD. Возможно измерение расстояния на карте (показан маршрут, проложенный пользователем). При необходимости можно найти в базе данных населенный пункт (по названию или части названия), просмотреть информацию по нему, и увидеть, где он расположен на карте (будет показан в центре карты и выделен цветом).

На рис. 34.7 показано дополнительное окно с информацией из базы данных, по объектам, попавшим в выделенный на карте контур.

Выделив объект в базе данных, можно увидеть его положение на местности.



Название	Муниц. образов.	Район	Название_2	Население	Площадь	Периметр	Описание
		Макушинский район		0.0	58058.390399933	991.42557541224	
Раздолье	Коноваловский/с/с	Макушинский район		0.17	360173.60910082	2673.8574225871	
Десятый Октябрь	Моршихинский/с/с	Макушинский район	0	0.14	395179.61425495	3003.2958286196	
Рекорд	Степновский с/с	Макушинский район	0	0.1	166768.5211401	2356.1101229591	
Журавлевка	Садоводский с/с	Макушинский район		0.27	515925.19019592	4080.0758284257	
Обутковское	Обутковский с/с	Макушинский район		0.74	944320.74794388	5577.3792766714	CC
Новая Роща	Пионерский с/с	Макушинский район	0	0.12	454989.4617033	3463.6616481017	
Садовод	Садоводский с/с	Макушинский район		0.38	440980.61764109	3018.1222171614	CC
Кленовый	Степновский с/с	Макушинский район		0.21	171393.22714806	2069.5132560058	
Степное	Степновский с/с	Макушинский район		0.5	595477.56864643	3568.9643800354	CC

Рис. 34.7. Просмотр информации объектов, выбранных на карте

Как это сделано

Мы привели пример разумного, на наш взгляд, сочетания различных технологий. Карта области была нарисована в трех идеологически разных программных продуктах — AutoCAD, ИнГЕО и Mapinfo в разное время и различными исполнителями. Пространственно координированные данные из разных источников были импортированы в базу данных Interbase.

Страница с картой размещена на Web-сервере **Tomcat**, работающем в операционной системе **Linux**. Доступ к базе данных осуществляется с помощью сервера баз данных **Firebird 1.5**, работающем под **Linux**, а пользователь просматривает карту с помощью **Avant Browser** (или любого другого, поддерживающего виртуальную машину Java) из операционной системы Windows (или иной, что не критично). На сервере вообще не лежит никаких графических файлов — только база данных, к которой могут подключаться столько пользователей, сколько выдержит "железо". Семантическую информацию в базе данных (например, численность населения по результатам переписи) редактируют ответственные специалисты, не мешая просмотру карты.

Изображение карты формируется в окне браузера на клиентской стороне с помощью локализированного нами Java-апплета **ALOV Map**[1]. Информация из базы данных передается порциями, при изменении зоны просмотра, и кэшируется у клиента.

Аналогичные карты могут быть сформированы и по территории любого населенного пункта или промышленного предприятия.

Все программные средства, выделенные здесь полужирным шрифтом, являются бесплатными (или практически бесплатными). Технические и экономические показатели демонстрируемого решения и линейки ГИС-продуктов от Autodesk читатели могут сравнить сами, ознакомившись с прайс-листами.

Как хранить координаты в базе данных

Итак, для того чтобы суметь нарисовать объекты независимо от применяемой программной среды, необходимо хранить координаты в базе данных. Чаще всего в ГИС применяются объекты трех типов:

- *точки* — положение объекта определяется парой координат X и Y (возможно и Z, и угол поворота);
- *линии* — геометрия объекта определяется списком координат вершин;
- *полигоны* — замкнутые контуры, геометрия объектов также определяется списком координат вершин.

Визуально объекты могут изображаться разными способами. В системе AutoCAD точечные объекты могут изображаться блоками, линейные — полилиниями, а полигональные — замкнутыми полилиниями со штриховками. В системе Mapinfo точечные объекты изображаются символами специальных картографических TrueType-шрифтов или растровыми изображениями, линейные — полилиниями, а полигональные — специальными замкнутыми областями.

Замечание

В отличие от AutoCAD в Mapinfo осуществляется автоматическая генерализация изображений — сохранение постоянных видимых размеров объектов при изменении масштаба изображения на экране. В системе AutoCAD слабым аналогом генерализации является управление весом линий. Конечно, и в AutoCAD можно запрограммировать автоматическое изменение масштабов блоков, ширины линий и густоты штриховок, но это будет связано с изменением свойств объектов в графической базе данных.

[1] **www.alov.org**.

Желательно избежать хранения внешнего вида объектов в базе данных, но если непременно требуется фиксированное изображение объекта, визуальные характеристики можно хранить в специальных полях. Сложность будет в том, что разные системы по-разному должны интерпретировать эти данные.

Существуют различные системы хранения пространственных данных, наиболее известной является Oracle SpatialWare. Безусловно, это очень мощный, надежный и универсальный инструмент, позволяющий решить все проблемы с хранением и использованием пространственных данных (при условии, если у вас нет финансовых проблем). Практически все ГИС-инструменты позволяют использовать это великолепное хранилище данных. По запросу "SpatialWare" поисковая машина Google выводит ссылки на тысячи страниц, на которых на разных языках расхваливаются достоинства продукта. По провокационному запросу "Кто использует SpatialWare" находится только четыре страницы — по случайному стечению обстоятельств это также страницы продавцов, а не потребителей. Процитируем одно из высказываний: "Когда мне требуется SpatialWare? Когда Вы имеете большие объемы изменяющихся пространственных данных, требующих доступа в реальном времени многим пользователям. SpatialWare также будет решением для случаев, когда нужен мощный пространственный анализ, а в клиентской части используются недостаточно производительные картографические системы".

С этим утверждением можно полностью согласиться. Но "большие объемы" данных, из-за которых вообще стоит применять такую мощную СУБД, как Oracle, могут образоваться в вашей ГИС только через несколько лет или вообще никогда не появиться. Так зачем же стрелять из пушки по воробьям, если можно использовать обычную рогатку?

На первых порах лучше использовать простые и дешевые решения ("рогатку"). Бесполезно сразу расходовать большие средства на столь "скоропортящиеся" продукты, как программы — когда их мощь действительно потребуется, купленные впрок версии наверняка устареют.

Организовать собственную систему хранения пространственных данных в СУБД совсем не сложно. Любой программист баз данных это может сделать. Опасность только в том, что без глубокого понимания сути задачи это можно сделать нерационально. После многих экспериментов мы пришли к выводу, что лучше всего координатную информацию хранить в одной таблице, в специальном MEMO-поле в виде единого списка всех координат вершин. Технические реализации хранения списка могут быть различными — мы работали с несколькими вариантами текстовых форматов (от Clipper-массива до LISP-списка), но теперь привели формат в соответствие с используемым в ALOV Map двоичным кодом. Переход на групповой стандарт и позволил применять продемонстрированные решения по публикации карт в Интернете.

Как создать собственную ГИС-систему

В рамках этой книги на такой вопрос ответить нельзя — на это можно ответить в специальной книге "ГИС на базе AutoCAD. Как это делается". Такая книга может появиться, если читатели проявят достаточную заинтересованность — ни авторы, ни издательство не знают, нужно ли это кому-нибудь.

Разрабатывая САПР, мы уже создали базовый фундамент и для разработки САПР-ориентированной ГИС.

У нас уже есть классификатор слоев, не столь, может быть, нужный в САПР, но чрезвычайно необходимый в ГИС. Мы разработали систему работы с электронными архивами проектов, калек и топографических планшетов, которая может использоваться и в ГИС *(см. главу 33)*. У нас имеется множество функций, облегчающих разработку программ любой сложности. Для наделения системы ГИС-функциями необходимо проработать следующие вопросы:

- поддержка дополнительных свойств слоев и структур данных в классификаторе;
- технология создания информационных объектов в AutoCAD;
- реализация хранения пространственных данных в СУБД;
- связывание базы данных с объектами рисунка;
- редактирование информационных объектов в базах данных и AutoCAD;
- комфортабельное редактирование баз данных;
- экспорт данных в AutoCAD и другие системы;
- получение координат объектов в базе данных из AutoCAD;
- просмотр объектов из базы данных в AutoCAD;
- управление видимостью информационных объектов;
- генерализация информационных объектов;
- организация топографического банка данных;
- системы разграфок топографических планшетов;
- манипуляции с топографическими планшетами.

Резюме

Завершая эту главу, в которой совсем не было "исходников", но было множество спорных рассуждений и утверждений, мы хотим пояснить следующее. Хотя на сей раз мы много критиковали продукты фирмы Autodesk, но делали это "по-родственному". Система AutoCAD и продукты на ее основе могут быть очень удобными инструментами для ГИС, прежде всего благодаря набору инструментов для разработчиков. Главным препятствием остается цена продуктов, терпимая в САПР, но неприемлемая для ГИС.

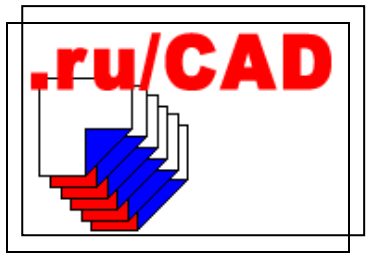
Если это обстоятельство для кого-то из читателей не является препятствием, то для разработки собственной ГИС на базе системы AutoCAD мы все-таки рекомендуем приобретать специализированные продукты (Autodesk Map в комплекте с Raster Design, как минимум). В любом случае вы будете иметь базовый AutoCAD, но вдобавок получите набор полезных утилит, многие из которых вам пригодятся. Autodesk Map имеет и собственный интерфейс для разработки прикладных программ.

Советуем обратить внимание на Raster Arts — серия программ, разработанных фирмой Consistent Software для работы с гибридной (растрово-векторной) графикой. По

нашему глубокому убеждению продукты российских разработчиков в этом секторе заведомо лучше импортных — хотя бы потому, что наши разработчики знакомы с достоинствами и недостатками "ихних", даже если не принимать во внимание, что "наши" вообще "чрезвычайно умны и сообразительны".

"Чрезвычайный ум" и "сообразительность" понадобятся также читателям, решившимся на внедрение ГИС-технологий — не для решения технических проблем, а для решения множества организационных и координационных задач. Это гораздо сложнее, чем "запрограммировать" ГИС на базе системы AutoCAD.

ЧАСТЬ V



Выпуск системы в свет

ГЛАВА 35

Создание справочной системы

Одним из основных отличий программного продукта от программы (или даже большой "кучи" программ, как у нас) является наличие документации и справочной системы. Это самое больное место большинства даже очень хороших программ[1]. Так уж "устроены" программисты, что заниматься скучной работой по разработке документации им очень не хочется. Документация всегда откладывается "на потом", а потом не остается времени. Мы в этом отношении не являемся исключением. Наши клятвы, произносившиеся в первых главах, мы хоть и не забыли, но и не выполнили. Полноценной справочной системы мы, разумеется, не имеем. Правда, одновременно с разработкой мы успели написать эту книгу, т. е., по традиционной манере российских программистов, попытались сделать "как лучше", хотя заказчик продукта нас об этом не просил.

В больших фирмах разработке документации уделяется огромное внимание — там этим занимаются специальные "дивизионы". Но даже в таких условиях наша любимая система AutoCAD и ее "десктопы" в последних версиях не имеют *полноценной* русскоязычной документации, какая была когда-то для AutoCAD R12.

В современных условиях бумажная документация все более сдает позиции. Многие производители стараются экономить и ограничиваются справочной системой и документацией в электронном виде. Конечно, настоящая бумажная документация удобней для пользователя. Очень многие пользователи до сих пор с благодарностью пользуются руководствами из упоминавшейся системы AutoCAD R12. Однако и культура многих пользователей в отношении использования справочных систем очень низка. Даже имея справки на родном языке, они ими не пользуются. Объясняется это, возможно, тем, что обычно справочные системы были англоязычными и малопонятными рядовым пользователям. Теперь это давно не так, но "наш человек" предпочитает потратить массу сил на поиск информации в Интернете, вместо того, чтобы нажать клавишу <F1> и решить вопрос за несколько секунд.

У нашей "фирмы" возможности по изготовлению документации минимальные, поэтому мы решаем обойтись электронными материалами. Пока. Пока не "разбогатеем". Ох уж эти благие намерения...

---

[1] Есть и приятные исключения — например, неоднократно упоминавшаяся нами программа ACADVar Алексея Васильченко.

Что уже сделано

Несмотря на покаянное и пессимистичное вступление, мы можем отметить, что у нас имеется хороший задел.

- *Во-первых*, при загрузке каждой LISP-программы может, по желанию пользователя, предварительно выводиться для просмотра файл с описанием работы этой программы. Именно для этой возможности мы и создавали "псевдопрограммы", фактически вызывающие одну функцию. Напомним, что справки к программам могут быть в виде простого текста или в HTML.
- *Во-вторых*, мы сделали так, чтобы этот файл справки, при запуске программы разработчиком, выводился в режиме редактирования. Тем самым мы заставляли себя написать сразу хоть несколько фраз о работе программы. Этого мало, но это уже кое-что.
- *В-третьих*, мы придумали такой же фокус с диалоговым окном **Советы дня**. Файлы советов у нас также понемногу накапливаются.
- *В-четвертых*, наша программа ruLispExplorer позволяет автоматически создавать HTML-файлы исходных текстов с подсветкой синтаксиса.
- *В-пятых*, у нас есть огромный резерв в виде пользователей-активистов, осваивающих нашу систему. В своих прошлых разработках мы применяли такую практику — предоставляли избранным людям в пожизненное бесплатное пользование наши программы в обмен на то, что они эти программы документируют. Есть категория людей, которым для того, чтобы что-то запомнить, нужно непременно законспектировать. Так пусть "конспектируют" функционирование нашей программы, изучая ее в работе. Таких людей мало, но пользу они приносят огромную.

HLP, CHM или HTML

Рано или поздно нам придется делать настоящую справочную систему. Текстовые "конспекты" нам, конечно, пригодятся, но придется сделать выбор формата справочников. Расписывать их особенности нашим читателям нет смысла — они с ними прекрасно знакомы. Первоначально в Windows использовались справочные HLP-файлы, в дальнейшем появился формат CHM. Оба формата имеют свои достоинства и недостатки. В системе AutoCAD до сих пор используются оба формата. Фирма Microsoft перспективным считает формат CHM, так оно, видимо, и будет.

С развитием Интернета весьма распространенным приемом стало использование справок в формате HTML. Действительно, очень удобно иметь одни и те же материалы на сайте и в справочной системе. Впрочем, работать на локальном компьютере с HTML-справками не очень-то удобно, т. к. нельзя воспользоваться поиском во всем содержимом. Современные инструментальные средства позволяют легко собрать из отдельных HTML-страниц единый CHM-файл.

Писать HTML, предназначенный для использования в справках, нужно как можно проще. Не следует применять никаких "украшений", все желательно делать в "академическом" стиле, с использованием минимальной номенклатуры логических тегов — заголовков, абзацев, списков, ссылок.

Инструмент для создания справок

Скомпоновать справочную систему с использованием современных инструментальных средств несложно. Не стоит даже рассчитывать на "официальную" технологию — написание текстов в формате RTF с последующей компиляцией. Систем разработки "хелпов" существует много. Выбор их определяется обычно финансовыми возможностями и привычками разработчиков.

Одной из лучших программ такого класса является Help & Manual (рис. 35.1).

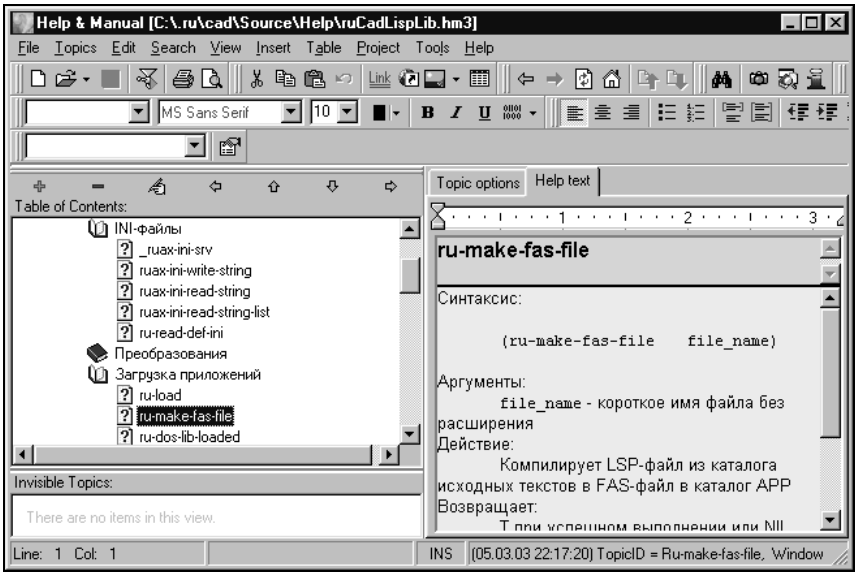


Рис. 35.1. Окно программы Help & Manual

Эта программа позволяет, используя один проект, изготовить справочную систему в форматах WinHelp (.HLP), HTML Help (.CHM), HTML, Adobe PDF, Microsoft Word (.RTF) и MultiMedia Help (e-Books). Очень полезна возможность создания проектов из HLP- и CHM-файлов с их декомпилированием.

Постепенное занесение в проекты Help & Manual наших текстовых и HTML-полуфабрикатов будет наилучшим решением. На технологии работы с этой программой мы не будем задерживаться. Она достойна написания специальной книги, хотя освоить ее можно самостоятельно за несколько часов.

Публикация в Интернете

Продвижение современных программ без собственного сайта невозможно. Тем более такой объемной системы, как наша. Для отдельных небольших программ можно обойтись их размещением в "софтовых" каталогах (и иметь от этого приличный доход), для крупной системы наличие собственного сайта обязательно. Бесплатный

хостинг на различных "народных" ресурсах вряд ли поможет, такие услуги предоставляются с ограниченным набором доступных средств и склонны, после собственной "раскрутки", превращаться в платные.

При создании сайта основное внимание нужно уделять не дизайну, а содержанию. Посетитель должен прежде всего получить полное представление о возможностях продукта и условиях приобретения. Вспомните, чего вам не удавалось обнаружить на сайтах, продвигающих программы, и не допускайте у себя таких ошибок.

О том, как создать сайт, мы, конечно, писать не будем[1]. Советами и руководствами по этой "кухне" заполнен Интернет.

Дополнительные справочники

Помимо справок по работе с программами в САПР можно включать множество справочной информации по различным аспектам проектирования. Каждый проектировщик работает, обложившись множеством справочников и нормативных документов. Как правило, в каждой книге "зачитаны до дыр" всего несколько страниц. Мы практически без дополнительных усилий можем дополнить свою систему мощной справочной службой, создавать которую будут сами пользователи.

Многие проектировщики, имеющие доступ к Интернету, давно "накачали" себе гигабайты всякого "добра" — от электронных изданий СНиП и ГОСТ до всяческих статей и невесть зачем прихваченных DWG-файлов. Но разобраться с этими драгоценными запасами и найти в них нужное бывает труднее, чем вновь разыскать материал в Интернете. Существуют и программы-каталогизаторы, хорошие и разные.

А ведь в нашей системе уже все есть для упорядочивания таких ресурсов. Мы умеем работать с XML-меню, можем встроить такое меню куда угодно (вспомните программу-стартер), можем описать в нем все, что угодно, да еще с иллюстрациями. Давайте предоставим пользователям ruCAD возможность самостоятельного ведения электронных справочников.

DWF-справка

Одним из возможных вариантов является создание справок в формате DWF. Это трудоемкий способ, т. к. предварительно нужно нарисовать требуемое в системе AutoCAD. Но такие "рисунки" всегда делались энтузиастами на бумаге, каких только "справок" не доводилось наблюдать. Научим нескольких дотошных "девочек" (увы, обычно бальзаковского возраста) этой нехитрой технологии, главный секрет которой — как написать XML-меню, мы много раз разбирали.

DWF-справки удобны тем, что внутри могут быть ссылки на другие ресурсы, в том числе на другие DWF. Самой мощной справочной системой такого рода, которую нам доводилось создавать, является топографический план г. Кургана М1:500 с общей схемой перекрытия города топографическими планшетами, возможностью просмотра любого из них и перехода на соседние. Специалистам, занимающимся проектированием генплана и инженерных коммуникаций, не нужно объяснять, что дает такая система в практической работе.

---

[1] Наш сайт, посвященный системе ruCAD, — **www.kurganobl.ru/cad**.

Растровые справочники

Во многих ситуациях поможет простое сканирование нужных страниц, сохранение в растровом формате и включение в XML-меню. Вспомните, наше XML-меню может иметь иллюстрацию к любому элементу. Обычно иллюстрации делаются маленькими по размеру, но ничто не мешает сделать их большими и основным содержанием, а прочие элементы — дополнительными комментариями.

Резюме

В этой очень короткой главе мы напомнили про некоторые возможности системы, позволяющие, в первых версиях, предоставить пользователям хотя бы суррогатную документацию. Идею о нагрузке пользователей черновой работой по разработке справок трудно реализовать, но это возможно. Это следует возложить на "дипломата", который обязательно должен быть в команде разработчиков.

ГЛАВА 36

Разработка инсталлятора

Любой программный продукт должен поставляться в виде установочного (инсталляционного) комплекта. С помощью инсталляционного комплекта пользователь с квалификацией "чайника"[1] должен иметь возможность установить программный продукт и при необходимости удалить его из системы. Некоторые программисты особо подчеркивают, как достоинство, что их программа не требует инсталляции, достаточно скопировать файлы в "подходящее" место. Как правило, это несложные в эксплуатации программы, которые действительно можно запускать из любого места, но при некоторых ограничениях — пользователю, от имени которого запускается программа, должно быть дозволено выполнять действия, выполняемые программой, например, запись файлов в те каталоги, в которые вздумает их писать эта программа.

Для серьезных программ такой способ установки (скопировал в любое место и запустил) просто невозможен. Мы уже писали о том, что система AutoCAD 2004 раскладывает свои файлы по сотням каталогов и производит запись в реестр нескольких тысяч ключей. Это делается не для того, чтобы попортить побольше крови пользователям и администраторам, а для того, чтобы обеспечить работу в современных многопользовательских операционных системах. Еще система AutoCAD 2002 устанавливалась так, что работать на ней мог только пользователь с правами Power User, что вызывало справедливый гнев системных администраторов, подрывало уважение к столь солидной фирме и заставляло искать лазейки для обхода ограничений.

Как работают инсталляционные программы, наши читатели, конечно, знают. Теперь нам предстоит узнать, как они делаются. Давайте разберем, что требуется проделать, чтобы установить систему ruCAD на компьютер конечного пользователя без программы-инсталлятора. Зная это, мы будем знать, *что* должен выполнить инсталлятор, а потом придумаем, *как* это сделать. Прежде всего вспомним основные вопросы, связанные с безопасностью в Windows NT/2000/XP (далее будем применять название Windows NT, как общего "предка" этих операционных систем). Заодно уточним и терминологию.

---

[1] На домашнем компьютере. Вряд ли в корпоративной системе администратор предоставит обычным пользователям право устанавливать программы.

Коротко о безопасности

Напомним, что в Windows NT каждый пользователь имеет два типа прав доступа: *разрешения* (permissions) и *права* (rights). Разрешение — возможность доступа к конкретному объекту определенным способом, например, записи в файл. Право — возможность выполнить действие, воздействующее на всю систему. Права и разрешения одновременно обычно называются *привилегиями* (privileges). Каждый пользователь при входе в Windows NT проходит процедуру регистрации[1], а при попытке доступа пользователя к любому объекту или свойству производится проверка привилегий данного пользователя и по результатам этой проверки доступ предоставляется или отклоняется. Каждому пользователю Windows NT соответствует учетная запись, определяемая именем пользователя и паролем, который он вводит при регистрации. Существуют две особые учетные записи с предопределенными привилегиями: *"Администратор"* (Administrator) и *"Гость"* (Guest). Для удобства управления пользователи объединяются в группы с одинаковыми привилегиями. При инсталляции Windows NT на рабочей станции создаются несколько стандартных предопределенных групп: *"Администраторы"* (Administrators), *"Опытные пользователи"* (Power Users), *"Пользователи"* (Users), *"Гости"* (Guests) и некоторые другие специальные группы. В доменных сетях существуют свои стандартные группы, разбор свойств которых не входит в рамки этой книги.

Замечание

В нашей системе также существуют пользователи, группы, администраторы и гости. Впредь, во избежание недоразумений, мы будем именовать их *пользователями ruCAD*, *группами ruCAD* и *администраторами ruCAD*.

Все члены группы "Администраторы", в которую по умолчанию входит пользователь "Администратор", обладают максимальными, из предоставляемых по умолчанию, правами[2] на управление системой — их много, а перечислять нет места. В контексте настоящей главы укажем только, что "Администраторы" могут взять на себя *владение* любым файлом. Члены группы "Опытные пользователи" не могут стать владельцами чужих файлов, но могут предоставлять в совместное использование папки. Члены группы "Пользователи" не могут предоставлять в совместное пользование каталоги (если только одновременно не являются членами групп "Опытные пользователи" или "Администраторы"). Обычно большинство пользователей являются членами группы — "Пользователи" и/или "Опытные пользователи".

Системный администратор может, а администратор домена просто обязан, создавать и иные группы пользователей с одинаковыми правами, например "Пользователи ruCAD". Каждый пользователь может входить в одну или несколько групп или вообще не входить в группы — это определяется политикой безопасности, проводимой администратором. Права членов стандартных групп могут быть изменены. Посколь-

---

[1] Регистрация может производиться автоматически, что не должно вводить в заблуждение — Windows NT в любом случае использует учетную запись пользователя для доступа к объектам от его имени. Автоматическая регистрация недопустима в корпоративной среде.

[2] Существуют права, не предоставляемые по умолчанию операционной системой Windows NT, даже администраторам, но это тема совершенно отдельного разговора.

ку представить и тем более разобрать все варианты управления правами не представляется возможным, мы ограничимся разбором только тех привилегий, которые имеют прямое отношение к установке и функционированию нашей системы. При упоминании стандартных групп и прав мы будем исходить из установок, производимых системой при инсталляции.

Обычно компьютер закреплен за одним "человеко-пользователем", но наведываться в гости могут и посторонние лица, в том числе незваные. Иногда на компьютере попеременно работают разные люди. Разумный пользователь, предоставляя компьютер даже "лучшей подруге", должен всегда выйти из системы и предоставить посетителю возможность зарегистрироваться под своим именем. Администратор, если ему это надо, доберется до ресурсов компьютера и в удаленном режиме.

В файловой системе NTFS, используемой в Windows NT, для каждого файла, помимо традиционных сведений (имени, размера, даты и времени последнего изменения), хранится также список управления доступом (Access Control List — ACL), определяющий для пользователей доступ к папкам и файлам. У каждого файла установлен *режим доступа* и *владелец* (как отдельный пользователь, так и группа). *Владелец* — это пользователь, создавший файл. Пользователи имеют разрешения (permissions) на действия с файлом, установленные операционной системой. Права на вновь созданный файл назначаются по правилам, установленным для контейнера (каталога), в котором он создается. Даже "Обычный пользователь" может запретить доступ к файлу, на который он имеет право на изменение доступа (Change Permission), всем пользователям, включая членов группы "Администраторы" и себя самого, но "Администратор" может взять на себя владение файлом (в журнале системных событий об этом останется запись), а "Опытный пользователь" "завладеть" чужим файлом, как правило, не может.

Как установить систему ruCAD вручную

Сценарий установки в значительной степени зависит от используемой операционной системы. Если бы система ruCAD применялась в Windows 98 или в Windows 2000/XP пользователем с правами администратора (что, само по себе, неоправданно), то все можно было бы проделать очень просто:

1. Скопировать (или разархивировать при поставке в виде ZIP-файла) каталог `.ru` со всеми подкаталогами в корень диска C:.
2. Выполнить командный файл c:\.ru\cad\bin\reg_ruSrv.bat для регистрации наших COM-серверов.
3. Выполнить файл c:\.ru\cad\bin\ruCAD.reg для внесения в реестр записей о расположении компонентов нашей системы.

Последние два действия можно было бы произвести с помощью одного командного файла. Если бы мы решили установить ruCAD в более подходящее место, например в каталог c:\Program Files\ru, то пришлось бы еще и отредактировать REG-файл.

Для запуска системы потребовалось бы создать ярлык файла ruCAD.exe на рабочем столе или в панели задач Windows. Прочие настройки (параметры пользователей, выбор рабочей системы AutoCAD) можно было бы осуществить из программы-стартера.

При установке ruCAD в Windows 2000/XP для работы обычного пользователя придется разложить файлы по папкам, доступным этому пользователю. Подробно эти вопросы мы рассматривали в *главе 3*.

Итак, программа-инсталлятор должна выполнить следующий минимальный набор операций:

- запросить каталоги для установки компонентов системы;
- скопировать в назначенные каталоги требуемые файлы;
- выполнить регистрацию компонентов системы в реестре и, при необходимости, произвести соответствующие записи в INI-файлах;
- запросить разрешение и, при положительном ответе, создать необходимые ярлыки в главном меню, на рабочем столе и в панели задач;
- обеспечить последующее корректное удаление системы.

Кроме того, инсталлятор обычно позволяет выбрать комплектность установки программы, а также выводит дополнительные сообщения — знакомит с лицензионным соглашением и особенностями программы. В нашей системе мог бы быть запрос нескольких каталогов, например для размещения классификатора слоев или базы данных спецификаций оборудования, которые могут уже существовать в общедоступных каталогах или даже на удаленном сервере. В рамках книжной версии мы этого делать не будем.

Присядем "на дорожку" и подумаем

А теперь задумаемся — все ли мы делали правильно до сих пор? В *главе 3* мы рассуждали о том, как будем сохранять настройки, но еще не знали, что именно будем сохранять. Теперь у нас имеются все программы, разработанные нами в соответствии с намеченной концепцией. До сих пор сами мы работали с привилегиями *Администратора* и делали что хотели. При этом мы считали, что наши решения обеспечат и работу *обычных пользователей*. Еще раз продумаем, верно ли это утверждение. Напомним, что права *пользователей ruCAD*, получаемые ими при регистрации в программе-стартере, не имеют никакого отношения к правам этих же людей, как *пользователей Windows*. Администратор САПР может быть "чайником" в работе с Windows и зарегистрирован в ней как обычный пользователь, хотя, конечно, такое нежелательно. Системный "суперадмин", в свою очередь, может ничего не понимать в САПР, что нежелательно еще более. Все дальнейшие рассуждения будут распространяться на *пользователей Windows NT/2000/XP*.

Кто должен устанавливать ruCAD

Устанавливать систему ruCAD должен будет администратор, а работать с ruCAD — обычный пользователь, или, правильнее, такой, который может работать с системой AutoCAD. С точки зрения работоспособности нашего программного продукта, установка системы администратором требуется из-за необходимости регистрации серверов (впрочем, это может делать и Power User) и потребности записи в раздел реестра `HKEY_USERS\.DEFAULT` (а это доступно только Администраторам).

Куда копировать файлы

Устанавливать файлы лучше всего в каталог, доступный всем пользователям. Хотя у нас, на период разработки, был предусмотрен каталог `Local Settings`, в нем не оказалось ничего, учитывающего специфику пользователя Windows. Для облегчения последующих настроек мы будем устанавливать все файлы в подкаталоги папки %ALLUSERSPROFILE%\Application Data\ruCAD. Внутри этой папки мы все-таки создадим подкаталоги:

- Shared — для файлов, которые могут переноситься в другие места, в том числе на разделяемые ресурсы других компьютеров;
- LocalData — для файлов, которые всегда должны находиться на компьютере пользователя;
- UserData — для специальных данных групп пользователей САПР;
- OtherData — для данных посторонних приложений, которые у нас непременно появятся.

Исполняемые файлы приложений будут устанавливаться в выбранный каталог, предлагая для этого %ProgramFiles%\ru\CAD[1].

Что и куда писать в реестр

Записи в реестре определяют расположение компонентов системы. Пока мы читали их из ветки `HKEY_CURRENT_USER\Software\ruCAD group\ruCAD`. Но как данные попадут в эту ветку для каждого пользователя, если устанавливает систему только один пользователь, хоть и администратор? Необходимо уточнить алгоритм работы.

При установке системы администратором производится запись в раздел реестра `HKEY_LOCAL_MACHINE\Software\ruCAD group\ruCAD`. Этот раздел доступен всем пользователям для чтения. При запуске стартера любым пользователем расположение компонентов определяется так:

- сначала ищется ключ в ветке `HKEY_CURRENT_USER\Software\ruCAD group\ruCAD`, если там обнаружены данные, то они используются, а иначе данные читаются из ветви `HKEY_LOCAL_MACHINE\Software\ruCAD group\ruCAD`;
- запись данных измененных пользователем производится в ветвь `HKEY_CURRENT_USER\Software\ruCAD group\ruCAD`;
- если после установки ruCAD будет создан новый пользователь Windows, то для него устанавливаются "настройки по умолчанию", копируемые из ветви `HKEY_USERS\.DEFAULT`. Мы также можем записать сюда некоторые данные, наподобие всяких "подлых ловушек", записываемых некоторыми программами.

Разборки с системой AutoCAD

Для того чтобы система ruCAD заработала сразу, необходимо, чтобы она знала, какую систему AutoCAD запускать, которая в свою очередь должна иметь профиль ruCAD, в котором в ключе с именем ACAD следует записать пути поиска наших

[1] Выделяем подкаталог ru\CAD потому, что у нас уже зреют "коварные замыслы" по созданию ru\GIS и ru\DOCS.

каталогов, да еще такие, куда система будет установлена. Такой профиль должен быть записан в ветви реестра

```
[HKEY_CURRENT_USER\Software\Autodesk\AutoCAD\R%AcadVersion%.0\%AcadKey%
\Profiles\ruCAD\General],
```

где `%AcadVersion%` — номер версии системы AutoCAD (15 или 16), а `%AcadKey%` — обозначение AutoCAD, соответствующее выбранному acad.exe.

Поскольку ветви `HKEY_CURRENT_USER` всех пользователей, кроме производящего установку, в момент инсталляции недоступны, создавать профиль также должна программа-стартер. Можно, конечно, обязать администратора создавать профили для каждого пользователя вручную, но администратор непременно забудет это сделать.

Не надо унывать!

Итак, хотя мы и старались предусмотреть при разработке программ все возможные ситуации, без их корректировки нам явно не обойтись. Вот тут и пригодятся дальновидные решения по инкапсуляции кода в одно место. Нам не придется корректировать десятки мест, в которых могло бы быть чтение данных из реестра, достаточно изменить два-три участка кода.

Как делаются установочные комплекты

Изготовить инсталляционный комплект можно даже с помощью архиватора RAR, позволяющего, при некотором навыке, подготовить довольно интеллектуальный сценарий установки программы, но обычно используются специализированные программные продукты, предназначенные для создания инсталляций. Все они имеют общий принцип работы:

- с помощью программы-оболочки[1] (обычно работающей в режиме Мастера) формируется сценарий установки программы;
- в соответствии со сценарием формируется дистрибутив продукта в виде одного или нескольких файлов, обычно с возможностью поставки на различных видах носителей.

При формировании дистрибутива в него могут включаться дополнительные библиотеки, требуемые для работы устанавливаемого продукта, причем некоторые инсталляторы позволяют, запустив приложение, подлежащее установке, выяснить, какие библиотеки оно использует, и включить их в комплект поставки. Мощные пакеты имеют в собственном составе архивы со всеми распространенными дополнительными компонентами (BDE, MDAC и т. п.).

Подготовка к созданию дистрибутива

Прежде чем формировать дистрибутив, необходимо как следует подготовить исходные файлы, из которых он будет комплектоваться. Наша система во время разработ-

[1] Некоторые инсталляторы не имеют интерфейсных оболочек, сценарий создается вручную, что иногда даже удобнее, а для отдельных оболочки разрабатывают сторонние авторы.

ки находится в каталоге c:\.ru\CAD. Сейчас у нас наверняка имеется масса мусора — устаревшие файлы, "натасканные про запас" блоки, документация, всякие посторонние утилиты и прочее неведомо откуда и зачем "притащенное добро". На всякий случай спрячем это все в специальный "архив-снимок" текущего состояния.

Уменьшение размеров приложений

Мы разработали довольно много приложений к ruCAD в среде программирования Borland Delphi и видим, что размер DLL- и EXE-файлов получается весьма значительный — от 0,4 до 1,4 Мбайт[1]. Если заглянуть в любой из таких файлов с помощью *подходящего* редактора ресурсов, то можно обнаружить внутри массу кнопок, картинок и текстовых строк, которые мы нигде не использовали. Вызвано это тем, что Delphi статически прикомпоновала к приложению пакеты, в которых содержатся стандартные ресурсы. Отделять используемые ресурсы от неиспользуемых компилятор и сборщик не умеют (таковы уж отрицательные последствия быстрой разработки).

Для приложения, состоящего из одного EXE-файла, включение внутрь всех его ресурсов может быть достоинством. Если файлов много и в каждом из них "зашита" одна и та же информация — это плохо.

Фирма Borland предусмотрела возможность использования *пакетов* (packages) — специальных динамически присоединяемых библиотек, содержащих визуальные компоненты и другие объекты. Приложение, откомпилированное с включенным флажком **Build with runtime packages** на вкладке **Packages** диалогового окна **Project Options**, будет иметь меньший размер. Вопрос в том, насколько меньший. Часто приводятся примеры, в которых *пустое* приложение, откомпилированное с поддержкой пакетов, "весит" в 20 раз меньше, чем при компилировании без пакетов, но при этом вместе с приложением размером 15 Кбайт нужно поставлять библиотеки общим размером до 5 Мбайт.

В реальных приложениях соотношение иное. Наша программа-стартер, откомпилированная с поддержкой пакетов, будет иметь размер 315 Кбайт, без поддержки — 860 Кбайт. Уменьшение размеров уже не в двадцать раз, но "головной боли" у нас будет больше, т. к. нам нужно *абсолютно точно* знать, какие именно дополнительные файлы мы должны будем поставлять в комплекте системы. Это будут не только стандартные библиотеки, но и пакеты времени выполнения из всех используемых нами дополнительных библиотек. Но и это еще полбеды. Если мы перекомпилируем хотя бы одно приложение в другой версии Delphi, то должны будем включить в поставку пакеты и из этой версии. Нет никаких гарантий, что во время всех этих "махинаций" с пакетами мы не допустим ошибок, за которые будут расплачиваться наши пользователи, которые в самый интересный момент могут получить сообщение об отсутствии какого-то файла. Поэтому мы принимаем решение — *не использовать пакеты времени исполнения.*

---

[1] Малый размер файлов, созданных в других средах, например в Visual Basic, не должен вводить в заблуждение — они также используют дополнительные ресурсы, только спрятанные в других DLL и OCX.

Перекомпиляция приложений

Далее следует перекомпилировать все Delphi-приложения. В каждом приложении следует снять отладочные режимы, убедиться, что флажок **Build with runtime packages** отключен, убрать все "черные ходы", которые мы оставляли, например, в диалогах регистрации пользователей, заполнить, где это доступно, информацию о версии программы. Кроме того, нужно проверить соответствие иконок приложений, логотипов и прочих оформительских элементов текущему состоянию нашей системы. Работа эта муторная, но больше откладывать ее нельзя.

Кстати, о логотипе

Внимательные читатели, возможно, заметили, что логотип и иконки приложений, показанные на нескольких рисунках, совсем не соответствуют рис. 2.2. Отвратительная (по нашему нынешнему мнению) "лапа" заменена на изящный (опять же по сегодняшнему представлению) "карандаш", составленный из нескольких "листов-слоев", имеющих цвета российского флага (цвет на черно-белых иллюстрациях не виден). Вызвано это тем, что рука сложной формы плохо масштабировалась на иконках, превращаясь в непонятное пятно. Пришлось тщательно, до пиксела, проработать один из наших прежних логотипов, сделать несколько вариантов для разных масштабов (иконки 16×16 и 32×32 пиксела, большие и малые заставки), включая эскиз рекламной растяжки, место для размещения которой мы уже наметили.

Упаковка файлов

Позаботиться об уменьшении размеров файлов не мешает. Нас не так волнует место, занимаемое программами на диске, как размер файлов дистрибутива, которые пользователям придется "качать" из Интернета. Есть еще одна возможность уменьшить размер EXE- и DLL-файлов — сжатие специальными программами. Часто применяется программа **UPX** (Ultimate Packer for executables)[1]. Это консольная утилита, позволяющая примерно в два раза снизить размер исполняемого файла. Для облегчения работы можно воспользоваться интерфейсной оболочкой к этой программе — **UpxShell**[2].

Использование UPX позволило снизить размер файлов в нашем каталоге BIN с 20 до 7,8 Мбайт. На размере дистрибутива это не скажется так значительно, т. к. файлы, уже сжатые UPX, будут в меньшей степени сжиматься при упаковке инсталлятором. В наших условиях размер дистрибутива уменьшается всего на 10%.

Однако все гораздо сложнее. Windows при запуске программы загружает в память не весь код сразу, а только часть кода, которая требуется для исполнения. Если программа имеет пункт меню **Print** и код для выполнения при выборе этого пункта, то код загружается в память только тогда, когда пользователь выбирает данный пункт. Если после загрузки кода в память он не используется некоторое время, то код выгружается из памяти, память освобождается и может быть использована другими программами (процесс, именуемый *paging*). При запуске нескольких экземпляров программы одна и та же область памяти используется для каждого из них.

"Компрессоры" EXE-файлов полностью блокируют работу метода *paging*, разжимая весь код в память и сохраняя его там до конца работы программы. Код в упакован-

---

[1] **upx.sourceforge.net**. Авторы Markus F.X.J. Oberhumer и Laszlo Molnar.

[2] **bash.hotbox.ru**. Автор Евгений Башкин.

ном EXE-файле хранится в таком формате, что операционная система не в состоянии разделять тот же блок памяти для нескольких экземпляров программы. В результате, сэкономив несколько мегабайтов на диске, мы можем получить перерасход куда более ценной оперативной памяти.

Эксперименты Джордана Рассела (Jordan Russell), автора программы **Inno Setup**, показали удивительные результаты: при сжатии файла MSACCESS.EXE с помощью UPX размер файла снизился на 2,2 Мбайт, но размер используемой памяти при запуске одного экземпляра программы увеличился на 5 Мбайт, а при запуске двадцати экземпляров — на 100 Мбайт по сравнению с несжатым файлом! Двадцать экземпляров мы запускать не будем, но дальнейшее баловство с "EXE-выжималками" прекратим.

Упаковка DWG-файлов

Не нужно забывать и об удалении "мусора" из DWG-файлов, входящих в поставку. Все файлы блоков и библиотек блоков следует очистить от неиспользуемых блоков, слоев и всего такого прочего с помощью команды PURGE или специальных утилит.

Второй этап чистки

После первого этапа "чистки рядов" скопируем все, как есть, в специальный каталог c:\.ru_ruInstall\cad, в котором всегда будем хранить исходные файлы, включаемые в дистрибутив, и начинаем более жестокую разборку, безжалостно удаляя все ненужные файлы.

Не забываем удалить все файлы с расширениями bak (временные файлы), todo (наши заметки для памяти), replace, blocks (временные файлы для переименования блоков), cdc, err, dmp, mnc, mns, mnr, Thumbs.db ("грязь" от AutoCAD), \cad\All Users\users.ini (история действий пользователей).

Разыскиваем и удаляем INI-файлы, в которых встречаются упоминания конкретных каталогов диска C: — FormStorage.ini, RU_USERS.INI.

Редактируем cad\Local Settings\Application Data\ruCAD\ru_users.mdb — удаляем всех пользователей, кроме **ADMIN** с паролем **rucad**.

В результате всех чисток общий размер наших файлов уменьшился с 88 до 43 Мбайт.

Для облегчения ориентировки в файлах перенесем их, как намечено, в подкаталоги Shared, LocalData, UsersData и OtherData каталога c:\.ru_ruInstall\cad.

Установка даты и времени

Обычно у всех файлов инсталляционного комплекта устанавливают единую дату и время, отражающие номер версии. Мы будем выпускать версию 0.9, и нам нужно установить время 00:09 и текущую дату выпуска. Сделать это можно разными способами, мы просто выделим в TotalCommander корневой каталог, выполним соответствующий пункт в меню **Файлы | Изменить атрибуты** (Files | Change Attributes) и установим параметры изменения атрибутов (рис. 36.1).

Теперь мы готовы к созданию установочного комплекта.

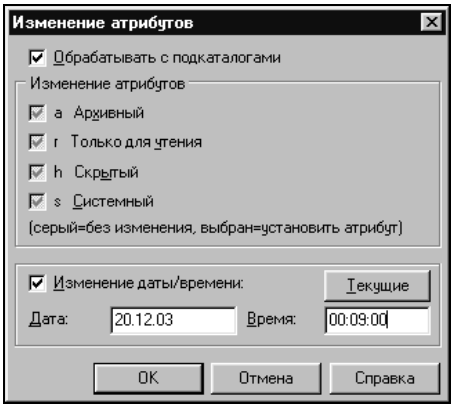


Рис. 36.1. Изменение даты и времени всех файлов комплекта

Выбор инструмента

Выбор продукта для создания инсталляций зависит от сложности требуемого сценария и комплектности системы. Немалое значение имеет и стоимость системы-инсталлятора. Очевидно, что нет смысла приобретать мощный инсталлятор **InstallShield Professional** стоимостью около $700 для создания дистрибутива условно-бесплатной программы, стоимостью $25. В то же время, если программа разрабатывается с использованием Delphi, то можно воспользоваться версией **InstallShield Express**, поставляемой с Delphi. Инсталляции отдельных приложений для AutoCAD можно создавать с помощью простой, но достаточно мощной программы **Ghost Installer** Алексея Попова. Эта программа, основанная на использовании архиватора RAR, теперь стала коммерческим продуктом, но имеет и бесплатную версию, отличающуюся тем, что сценарий нужно писать вручную. Когда-то автор учел некоторые наши советы по совершенствованию программы, а мы ее использовали для создания инсталляций системы BestIA.

Помимо упомянутых систем подготовки инсталляций используются и другие превосходные продукты (MindVision's Installer VISE, WISE Installer, GkSetup, CreateInstall и др.). Мы их исследовать не будем, а воспользуемся хорошим и бесплатным инсталлятором **Inno Setup**[1]. Этот инсталлятор имеет достаточный ассортимент возможностей, доступен для бесплатного использования с исходными текстами и имеет несколько графических оболочек разных авторов.

Работа с программой Inno Setup

Внешне основная программа Inno Setup (Compil32.exe) представляет собой простенький, казалось бы, текстовый редактор сценариев инсталляции (рис. 36.2), из которого можно запустить сценарий и скомпилировать дистрибутив.

[1] **www.jrsoftware.org**.

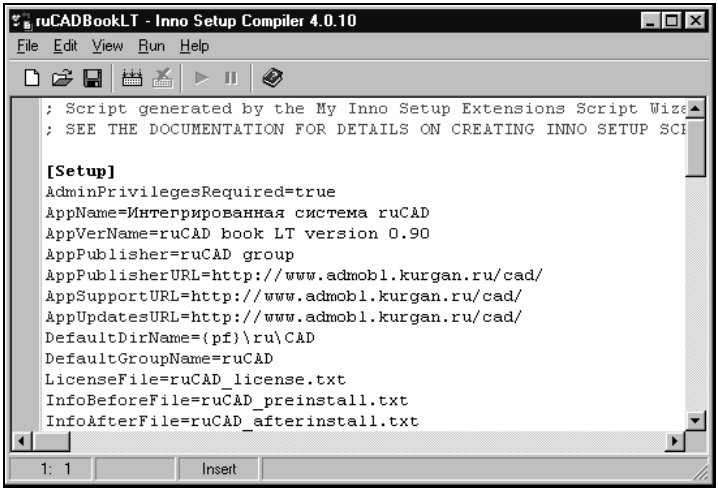


Рис. 36.2. Программа Inno Setup Compiler

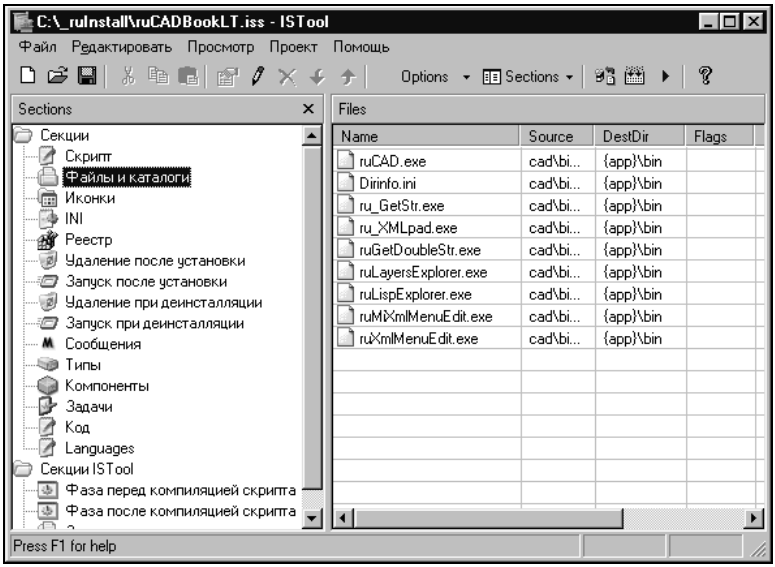


Рис. 36.3. Окно программы ISTool

Мощь и возможности этой замечательной программы заключаются в том, как она обрабатывает сценарий и что в этом сценарии можно предусмотреть. Программа Inno Setup имеет массу возможностей, описание которых заняло бы очень много места. То, что потребуется нам, мы разберем на конкретном сценарии установки системы ruCAD. Файл сценария можно редактировать и вручную, но для этого нужно знать массу мелких деталей, касающихся различных параметров, условий их применения и условных обозначений в сценарии. Когда-то мы их будем хорошо знать, а сначала лучше воспользоваться специальной программой-оболочкой

ISTool[1]. Эта программа (рис. 36.3) облегчает разработку сценария. Разработчик имеет удобный интерфейс, позволяющий как редактировать сценарий в виде текста, так и редактировать любой элемент сценария с использованием специальных форм, выбором допустимых вариантов и параметров в диалоговых окнах. Весьма разумное сочетание "визуальности" и редактирования текста, в котором отображаются результаты интерактивных действий, позволяет быстро освоить подготовку сценариев для программы Inno Setup.

Мы не будем разбирать работу с программами — она очень проста. Детальному рассмотрению мы подвергнем наш сценарий инсталляции.

Сценарий инсталляции

Сценарий инсталляции сохраняется в файле с расширением iss. Формат файла сценария напоминает формат INI-файлов, но есть и некоторые отличия — не всегда имя переменной отделяется от значения символом "=". Запустив программу ISTool, мы, с помощью Мастера, создаем файл c:\.ru_ruInstall\ruCADBookLT.iss, в котором вначале имеются заготовки основных параметров будущей инсталляции. Далее мы, иногда вручную, иногда с помощью диалоговых окон (рис. 36.4), редактируем сценарий.

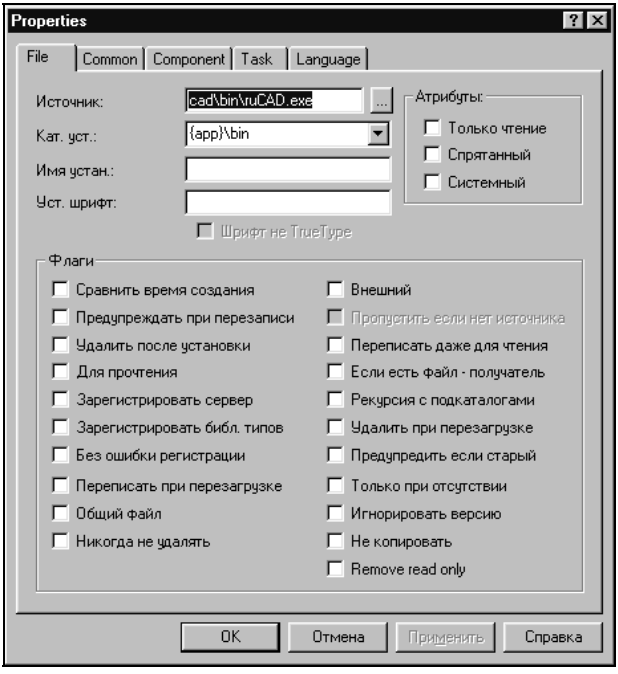


Рис. 36.4. Диалоговое окно свойств элемента

[1] **www.istool.org**.

Замечание

Далее мы будем приводить усложняющиеся фрагменты сценария в отдельных листингах (по секциям). Для улучшения восприятия мы будем переносить длинные строки, добавленные нами пространные комментарии (которых в реальном сценарии нет) мы будем писать курсивом под чертой, введенные нами данные — выделять полужирным шрифтом. Ключевые слова программы Inno Setup мы будем выделять подчеркиванием.

В сценарии часто используются символьные *константы*, указанные в фигурных скобках. Во время выполнения программы установки на компьютере пользователя вместо констант подставляются их значения, действующие на компьютере пользователя. Например, константа **{localappdata}** на одном компьютере (в Windows 98) будет соответствовать каталогу c:\Windows\Local Settings\Application Data, а на другом (в Windows XP) — c:\Documents and Settings\ShaggyDoc\Local Settings\Application Data. Константа **{commonappdata}** будет соответствовать (в Windows XP) — c:\Documents and Settings\All Users\Application Data. Смысл констант мы будем разъяснять в комментариях. Сценарий формируется для установки на русском языке, хотя имеется возможность выбора языка инсталляции. В этом случае потребовались бы варианты языкозависимых параметров, например, разных файлов с текстом лицензии.

Основные параметры приложения занесены в секцию `Setup` (листинг 36.1).

Листинг 36.1. Файл сценария ruCADBookLT.iss. Секция `Setup`

```
[Setup]
;-----------------------------------------------------------------------
; Наименование нашего приложения и основные параметры установки
;-----------------------------------------------------------------------
AppName=Интегрированная система ruCAD
AppVerName=ruCAD book LT version 0.90
AppPublisher=ruCAD group
AppPublisherURL=http://www.admobl.kurgan.ru/cad/
AppSupportURL=http://www.admobl.kurgan.ru/cad/
AppUpdatesURL=http://www.admobl.kurgan.ru/cad/
;-----------------------------------------------------------------------
; Куда, по умолчанию, будем предлагать установить систему
; Константа {pf} обычно соответствует c:\Program Files
;-----------------------------------------------------------------------
DefaultDirName={pf}\ru\CAD
DefaultGroupName=ruCAD
;-----------------------------------------------------------------------
; Файл с текстом лицензионного соглашения. Указываем один вариант -
; только на русском языке. О содержании лицензии см. главу 37
;-----------------------------------------------------------------------
LicenseFile=license.txt
;-----------------------------------------------------------------------
; Файл с текстом, выводящимся перед установкой
;-----------------------------------------------------------------------
InfoBeforeFile=preinstall.txt
```

```
;-------------------------------------------------------------------
; Файл с текстом, выводящимся после установки
;-------------------------------------------------------------------
InfoAfterFile=readme.txt
;-------------------------------------------------------------------
; Имя EXE-файла инсталляции. Задаем ruCADSetup.exe
;-------------------------------------------------------------------
OutputBaseFilename=ruCadSetup
;-------------------------------------------------------------------
; Минимальная версия Windows. Задана Windows 98 до Second Edition
;-------------------------------------------------------------------
MinVersion=4.1.1998,4.0.1381
AppCopyright=ruCAD group
;-------------------------------------------------------------------
; Требуемые привилегии для запуска ruCADSetup.exe. Указываем
; константу admin, т. к. потребуется регистрация COM-серверов
; Если не задать ограничение, сбой произойдет слишком поздно
;-------------------------------------------------------------------
PrivilegesRequired=admin
;-------------------------------------------------------------------
; Отключение вывода страницы о пользователе Windows. Незачем
;-------------------------------------------------------------------
UserInfoPage=false
AlwaysShowDirOnReadyPage=true
AlwaysShowGroupOnReadyPage=true
;-------------------------------------------------------------------
; Вывод диалога для выбора языка инсталляции
;-------------------------------------------------------------------
ShowLanguageDialog=auto
;-------------------------------------------------------------------
; Задаем картинку, выводящуюся в левой стороне мастера. Это должен
; быть файл формата BMP размером 164x314 пикселов. Можно не задавать,
; тогда будет выведена стандартная картинка
;-------------------------------------------------------------------
WizardImageFile=C:\.ru\_ruInstall\ruInno164_314.bmp
;-------------------------------------------------------------------
; Задаем картинку, выводящуюся в шапке Мастера. Это должен
; быть файл формата BMP размером 55x55 пикселов. Можно не задавать,
; тогда будет выведена стандартная картинка
;-------------------------------------------------------------------
WizardSmallImageFile=C:\.ru\_ruInstall\ruInno55.bmp
```

В листинге 36.2 приведена секция `Tasks` сценария. В этой секции можно описать несколько дополнительных задач. Мы оставляем одну, предлагаемую по умолчанию, задачу "Создать ярлык на рабочем столе" и добавляем свою задачу "Установить DOSLib".

Листинг 36.2. Файл сценария ruCADBookLT.iss. Секция `Tasks`

```
[Tasks]
Name: desktopicon; Description: Создать ярлык на рабочем столе;
        GroupDescription: Дополнительные значки:
Name: doslib; Description: Установить DOSLib;
        GroupDescription: Дополнительные приложения:
```

Далее формируем важнейшую секцию сценария — `Files` (листинг 36.3). Именно в этой секции задают, какие файлы нужно включать в установочный комплект и куда эти файлы должны устанавливаться на компьютере пользователя. Файлы можно перечислять как по одному, так и по маске. Для каждого файла могут быть установлены индивидуальные параметры установки. Мы не будем разбирать все параметры, прокомментируем только используемые в нашем сценарии. Для каждого описания обязательным является параметр `Source` (расположение файла относительно сценария) и `DestDir` (размещение этого файла на компьютере пользователя). При указании `DestDir` следует использовать константы, обозначающие физические имена каталогов, вычисляемые во время установки.

Нам предстоит установить несколько тысяч файлов. Описывать каждый отдельный файл в сценарии мы, разумеется, не будем. Нам поможет заранее продуманная *(см. главу 3)* система каталогов, позволяющая установить большие группы файлов в любые места с учетом привилегий пользователя. Файлы из каталога Bin мы будем описывать индивидуально, т. к. для некоторых из них требуется регистрация в качестве COM-сервера, а для некоторых — нет. В листинге мы приведем только несколько примеров. Описание каждого файла начинается с ключевого слова `Source` и выполняется в виде одной строки, но в листинге мы сделаем разбивки на строки по параметрам. Сделано это из-за формата издания и для улучшения восприятия.

Ключевое слово `Components` указывает на принадлежность файла к группе, определенной в секции `Components` (см. листинг 36.9). Создав в сценарии несколько групп, можно предусмотреть различные варианты установки (полная, минимальная, выборочная и т. п.). Мы (несколько искусственно) предусмотрели группы компонентов для проверки возможностей инсталлятора. По мере развития системы мы можем предусмотреть разбивку на более логичные группы, например, по специальностям (АР, ОВ, ВК). Ограничений здесь нет, вопрос только в том, как выделить необходимые только для этой группы файлы — слишком "переплетены" все наши функции.

Листинг 36.3. Файл сценария ruCADBookLT.iss. Секция `Files`

```
[Files]
;-----------------------------------------------------------------
; Таким образом записываются ActiveX DLL. Включаются в обязательную
; группу компонентов main, указывается необходимость регистрации
;-----------------------------------------------------------------
Source: cad\bin\AxAcadStatusBarButton.dll;
   DestDir:   {app}\bin;
   Components:  main;
   Flags:    regserver
```

```
;-------------------------------------------------------------------
; Аналогично включаем и другие DLL и OCX, требующие регистрации
;-------------------------------------------------------------------
Source: cad\bin\ruADOConnSvr.dll;
   DestDir: {app}\bin;
   Components: main;
   Flags: regserver

Source: cad\bin\AcadColor.ocx;
   DestDir: {app}\bin;
   Components: main;
   Flags: regserver

;-------------------------------------------------------------------
; А это обычная DLL, не требующая регистрации
;-------------------------------------------------------------------
Source: cad\bin\rudwginfo.dll;
   DestDir: {app}\bin; Components: main
;-------------------------------------------------------------------
; Далее перечислены EXE-файлы. Для некоторых из них может быть
; указана задача - desktopicon, позволяющая создать для этого файла
; значок на рабочем столе. Нам достаточно создать значок для
; программы-стартера
;-------------------------------------------------------------------
Source: cad\bin\ruCAD.exe;
   DestDir: {app}\bin;
   Components: main;
   Tasks: desktopicon
Source: cad\bin\ruLayersExplorer.exe;
   DestDir: {app}\bin;
   Components: main;
;-------------------------------------------------------------------
; Dirinfo.ini нужно просто скопировать
;-------------------------------------------------------------------
Source: cad\bin\DIRINFO.INI; DestDir: {app}\bin; Components: main

;-------------------------------------------------------------------
; Далее записаны большие группы каталогов. Для файлов указана маска *,
; каталоги назначения заданы константами, указаны группы компонентов
; и Flags: recursesubdirs, указывающий на необходимость обработки
; всех подкаталогов. Показаны, для примера, только несколько каталогов
;-------------------------------------------------------------------
Source: cad\Shared\app\*;
  DestDir: {commonappdata}\ruCAD\Shared\app;
  Components: main;
  Flags: recursesubdirs
```

```
;-----------------------------------------------------------------------
; Файлы справок входят в группы main и help
;-----------------------------------------------------------------------
Source: cad\Shared\help\*;
  DestDir: {commonappdata}\ruCAD\Shared\help;
  Components: main help; Flags: recursesubdirs
;-----------------------------------------------------------------------
; HTML-файлы входят в группы main и html
;-----------------------------------------------------------------------
Source: cad\Shared\html\*;
  DestDir: {commonappdata}\ruCAD\Shared\html;
  Components: main html; Flags: recursesubdirs
;-----------------------------------------------------------------------
; Файлы классификатора слоев входят в группы main и class
;-----------------------------------------------------------------------
Source: cad\Shared\Layers\*;
  DestDir: {commonappdata}\ruCAD\Shared\Layers;
  Components: main class; Flags: recursesubdirs
;-----------------------------------------------------------------------
; Файлы типовых текстов входят в группы main и txt
;-----------------------------------------------------------------------
Source: cad\Shared\Txt\*;
  DestDir: {commonappdata}\ruCAD\Shared\Txt;
  Components: main txt; Flags: recursesubdirs
;-----------------------------------------------------------------------
; А эти файлы должны устанавливаться в папку localdata
;-----------------------------------------------------------------------
Source: cad\LocalData\*;
   DestDir: {commonappdata}\ruCAD\LocalData;
   Components: main; Flags: recursesubdirs
;-----------------------------------------------------------------------
; Файлы, копируемые в корневой каталог приложения
;-----------------------------------------------------------------------
Source: license.txt; DestDir: {app}; Components: main
Source: readme.txt; DestDir: {app}; Components: main
```

В секции `Icons` (листинг 36.4) указываем, какие создавать значки и группы меню.

Листинг 36.4. Файл сценария ruCADBookLT.iss. Секция `Icons`

```
[Icons]
;-----------------------------------------------------------------------
; Создаем группу в главном меню
;-----------------------------------------------------------------------
Name: {commonprograms}\Интегрированная система ruCAD;
      Filename: {app}\bin\ruCAD.exe;
      Comment: Стартер ruCAD
```

```
;-----------------------------------------------------------------------
; Даем указание создать значок на рабочем столе каждого пользователя
;-----------------------------------------------------------------------
Name: {commondesktop}\Интегрированная система ruCAD
  ; Filename: {app}\bin\ruCAD.exe; Tasks: desktopicon
```

Секцию `Run` (листинг 36.5) мы оставляем пустой. Здесь мы могли бы указать программы, запускаемые до и после установки системы.

Листинг 36.5. Файл сценария ruCADBookLT.iss. Секция `Run`

```
[Run]
; NOTE: The following entry contains an English phrase ("Launch"). You are free to
translate it into another language if required.
```

Секцию `Dirs` (листинг 36.6) мы также оставляем пустой. Здесь мы могли бы указать, какие каталоги (помимо автоматически создаваемых при копировании файлов) требуется создать или удалить. Обычно, таким образом, некоторые программы создают пустые каталоги для собственных данных. Мы предпочитаем, чтобы это делали сами пользователи, по своему усмотрению.

Листинг 36.6. Файл сценария ruCADBookLT.iss. Секция `Dirs`

```
[Dirs]
```

В важнейшей секции `Registry` (листинг 36.7) записываются требуемые ключи. Напомним, что для "внутреннего употребления" у нас имеется REG-файл, но у пользователя ключи в реестре должны быть созданы автоматически. В значениях ключей записываются каталоги, в которые установлена система. Добавлен флаг **uninsdeletekey**, указывающий на необходимость удаления ключа реестра при деинсталляции системы. Мы пока не предусмотрели выбор рабочей системы AutoCAD и запись информации о ней в реестр. Это все равно сделает программа-стартер при первом запуске. В финальных версиях системы ruCAD мы предусмотрим все ситуации. До запуска основного процесса установки будет запускаться специальная функция, проверяющая наличие системы AutoCAD на компьютере, позволяющая выбрать его версию и разрешающая или запрещающая основной процесс установки.

Листинг 36.7. Файл сценария ruCADBookLT.iss. Секция `Registry`

```
[Registry]
;-----------------------------------------------------------------------
; Несколько примеров записи в реестр
;-----------------------------------------------------------------------
Root: HKLM; SubKey: SOFTWARE\ruCAD group\ruCAD;
ValueType: string; ValueName: RootDir;
ValueData: {app};
Flags: uninsdeletekey; Components: main
```

```
Root: HKLM; SubKey: SOFTWARE\ruCAD group\ruCAD;
ValueType: string; ValueName: AppDataDir;
ValueData: {commonappdata}\ruCAD\Shared;
Flags: uninsdeletekey; Components: main

Root: HKLM; SubKey: SOFTWARE\ruCAD group\ruCAD;
 ValueType: string; ValueName: LocalSettingsDir;
ValueData: {localappdata}\ruCAD\LocalData;
Flags: uninsdeletekey; Components: main

Root: HKLM; SubKey: SOFTWARE\ruCAD group\ruCAD;
ValueType: string; ValueName: UserConnectionString;
ValueData: "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source={commonappdata}\ruCAD\LocalData\ru_users.mdb;Persist Security Info=False";
Flags: uninsdeletekey; Components: main
```

В секции `ISTool` (листинг 36.8) записаны некоторые параметры самого инструмента разработки сценария — разрешено использование сжатия по алгоритму 7zip[1] и указан LOG-файл, в который будут записываться сообщения компилятора сценария.

Листинг 36.8. Файл сценария ruCADBookLT.iss. Секция `ISTool`

```
[_ISTool]
Use7zip=True
LogFile=ruCAD_Install.LOG
LogFileAppend=false
```

В секции `Components` (листинг 36.9) описаны группы компонентов, на которые мы ссылались при формировании предыдущих секций.

Листинг 36.9. Файл сценария ruCADBookLT.iss. Секция `Components`

```
[Components]
;-------------------------------------------------------------------
; Группа main, видимая пользователю как "Основной комплект".
; Имеет флаг fixed, указывающий, что эту группу компонентов нельзя
; отключить при установке. Входит в типы Custom, Compact и Full,
; т. е. в "Выборочную", "Компактную" и "Полную" установки
;-------------------------------------------------------------------
Name: main; Description: Основной комплект;
  Flags: fixed; Types: custom compact full
;-------------------------------------------------------------------
; Группа help, видимая пользователю как "Справки"
; Не имеет флага fixed, т. е. эти компоненты можно не ставить
```

[1] Включение этой опции приводит к тому, что сама программа установки будет помещена в самораспаковывающийся архив. Размер файла при этом уменьшится почти в два раза, но потребуется время на автоматическую распаковку. На компьютере пользователя это займет несколько секунд, при сборке установочного комплекта — от нескольких минут до часов (в зависимости от производительности компьютера разработчика).

```
; Входит в типы Custom и Full,
; т. е. в "Выборочную" и "Полную" установки
;-----------------------------------------------------------------------
Name: help; Description: Справки; Types: custom full
;-----------------------------------------------------------------------
; Группа class, видимая пользователю как "Классификатор слоев"
; Не имеет флага fixed, т. е. эти компоненты можно не ставить
; Входит в типы Custom и Full,
; т. е. в "Выборочную" и "Полную" установки
;-----------------------------------------------------------------------
Name: class; Description: Классификатор слоев; Types: custom full
;-----------------------------------------------------------------------
; Группа html, видимая пользователю как "HTML-справки"
; Не имеет флага fixed, т. е. эти компоненты можно не ставить
; Входит в типы Custom и Full,
; т. е. в "Выборочную" и "Полную" установки
;-----------------------------------------------------------------------
Name: html; Description: HTML-справки; Types: custom full
;-----------------------------------------------------------------------
; Группа txt, видимая пользователю как "Типовые тексты"
; Не имеет флага fixed, т. е. эти компоненты можно не ставить
; Входит в типы Custom и Full,
; т. е. в "Выборочную" и "Полную" установки
;-----------------------------------------------------------------------
Name: txt; Description: Типовые тексты; Types: custom full
```

В секции Languages (листинг 36.10) записаны некоторые параметры языка установочного комплекта. Мы предусмотрели один язык, но можно описать несколько языков для инсталляционной программы. Для каждого языка можно задать имена файлов сообщений программы, лицензионного соглашения, текстов, выводимых до и после работы программы установки. Кстати, для любого из ранее описанных компонентов можно указать языковую опцию, т. е. при выборе какого языка устанавливать этот компонент.

Листинг 36.10. Файл сценария ruCADBookLT.iss. Секция Languages

```
[Languages]
Name: Russian;
 MessagesFile: C:\Program Files\Inno Setup 4\Russian-19-4.0.0.isl;
 LicenseFile: C:\.ru\_ruInstall\license.txt;
 InfoBeforeFile: C:\.ru\_ruInstall\preinstall.txt;
 InfoAfterFile: C:\.ru\_ruInstall\readme.txt
```

Как найти систему AutoCAD

Установка системы ruCAD имеет смысл только в том случае, если на компьютере установлена система AutoCAD. Поиском AutoCAD из программы-стартера мы занимались в *главе 20*. Теперь это нужно проделать до установки ruCAD. Более того, нам

необходимо для выбранной системы AutoCAD создать профиль ruCAD и записать в пути поиска наши каталоги, да еще такие, куда система будет установлена.

Начнем решение с конца. Профиль должен быть создан в ключе реестра. Из множества параметров профиля нас интересует только ветка

```
[HKCU\Software\Autodesk\AutoCAD\R% AcadVersion %.0\%AcadKey%\Profiles\ruCAD \General]
```

и ключ `ACAD`, в начало которого мы должны вписать свои каталоги. Сделать это можно множеством способов, но мы постараемся найти простой. Вспомним, что программа-стартер позволяла выбрать систему AutoCAD из списка установленных путем вызова функции `ruSelectAcad`. Эта функция, после выбора пользователя, сразу записывала в реестр параметры выбранной системы AutoCAD, в частности `AcadKey` и `AcadVersion`, т. е. то, что нам надо. Итак, нам необходимо в начале инсталляции вызвать это диалоговое окно, если выход из него будет произведен по кнопке **Отмена** — прервать инсталляцию, а если выход будет произведен по кнопке **OK** — прочитать данные из реестра и вписать в профиль. Все это легко проделать в своей программе, но как это сделать при инсталляции?

В сценарии программы Inno Setup может быть очень интересная секция `Code`, в которую можно включить собственные процедуры и функции, написанные на Pascal, вызывать функции из собственных DLL, включать собственные диалоги.

Сначала мы напишем на Delphi небольшую динамическую библиотеку DLL (листинг 36.11). Функция `SelectAcad` выводит диалоговое окно выбора системы AutoCAD и записывает результаты в реестр в ветвь `HKEY_CURRENT_USER` или `HKEY_LOCAL_MACHINE`.

Листинг 36.11. Файл ruSelAcad.dpr

```
library ruSelAcad;

uses
  SysUtils,Windows, Classes, frmSelectAcad, ruUtils,  JclRegistry, JclStrings;

{$R *.RES}

function SelectAcad(): boolean; stdcall;
begin
  result := ruSelectAcad;
end;

exports
  SelectAcad;

begin
end.
```

Откомпилируем проект и полученный файл ruSelAcad.dll сохраним под именем c:\.ru_ruInstall\cad\AddOn\ruSelAcad.dll. Добавим в секцию `Files` (листинг 36.12) еще одну строку, включающую DLL в инсталляцию, но не устанавливающую на компьютер пользователя — флаг **dontcopy**.

Листинг 36.12. Файл сценария ruCADBookLT.iss. Секция `Files`

```
Source: cad\AddOn\ruSelAcad.dll; Flags: dontcopy
```

В сценарий установки добавим секцию `Code` (листинг 36.13). В этой секции мы объявили собственную функцию `SelectAcad`, заключенную в файле ruSelAcad.dll, и переопределили стандартную функцию программы Inno Setup `NextButtonClick`. В результате инсталляция будет запускаться, но после просмотра информационного сообщения (в котором мы напишем о необходимости выбора системы AutoCAD) при нажатии кнопки **Далее** будет появляться диалоговое окно выбора AutoCAD (см. рис. 20.3). При закрытии этого окна кнопкой **Отмена** будет выдано сообщение об ошибке с предложением отменить инсталляцию. Таким образом или сведения о системе AutoCAD будут записаны в реестр, или установка будет отменена.

Листинг 36.13. Файл сценария ruCADBookLT.iss. Секция `Code`

```
function SelectAcad():boolean;
 external 'SelectAcad@files:ruSelAcad.dll stdcall';

function NextButtonClick(CurPage: Integer): Boolean;
begin
  Result := True;
  if CurPage = wpInfoBefore then Result:= SelectAcad;
  if not result then
   MsgBox('Не выбран рабочий AutoCAD!'+#13+
  'Отмените установку или повторите выбор', mbCriticalError,MB_OK);
 end;
```

Сборка установочного комплекта

После подготовки сценария в среде ISTools можно откомпилировать (<Ctrl>+<F9>) или откомпилировать и запустить (<Ctrl>+<F5>) установку (рис. 36.5 и 36.6). Если в сценарии нет ошибок, то по истечении времени, необходимого на обработку сценария и упаковку файлов, будет создан файл дистрибутива C:\.ru_ruInstall\Output \ruCadSetup.exe.

Испытания установочного комплекта

Испытаниям и совершенствованию полученного "изделия" теперь можно посвятить всю оставшуюся жизнь. Но начать следует с проверки того, как происходит сам процесс инсталляции, а потом уже — как работает установленная система. Производить испытания нужно на разных компьютерах, под всеми операционными системами, на которые мы ориентированы (минимум Windows 98, максимум, пока, Windows XP). Прежде всего проверим, туда ли, куда мы задумывали, устанавливается наша система.

При установке по подготовленному сценарию компоненты системы установились в перечисленные ниже каталоги.

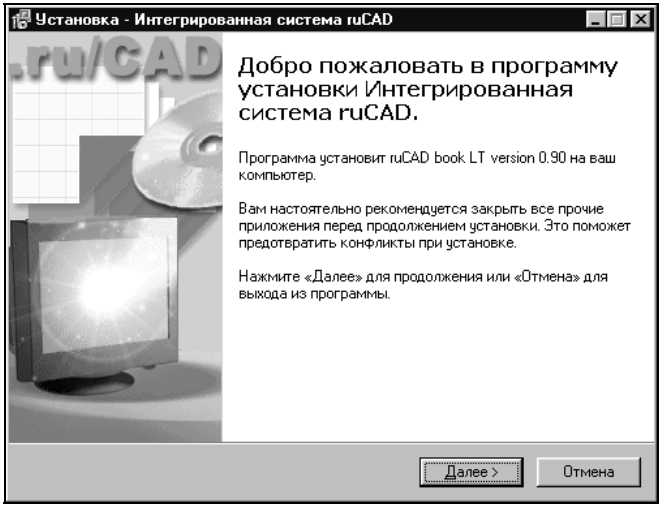


Рис. 36.5. Первая страница Мастера установки

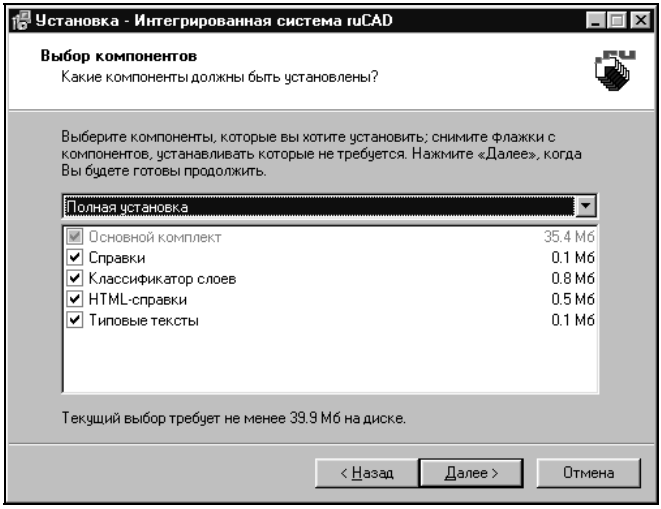


Рис. 36.6. Страница выбора компонентов Мастера установки

При установке под Windows 98:

- в каталог c:\Program Files\ru\CAD\bin — EXE-файлы и DLL;
- в каталог c:\Windows\Application Data\ruCAD\Shared и его подкаталоги (app, block, block-lib, block-lw, dcl, help, html, Layers) — файлы, общие для всех пользователей;
- в каталог c:\Windows\Application Data\ruCAD\LocalData и его подкаталоги — файлы для настроек конечного пользователя.

Конечно, это не очень удобно. В Windows 98 мы могли бы все свалить в одну большую кучу в подкаталоги c:\Program Files\ru\CAD, для этого надо было бы написать более "хитрый" сценарий. Но рядом с нашими каталогами расположился и c:\Windows\Local Settings\Application Data\Autodesk\AutoCAD 2004, а это уже "подходящая компания", и свидетельство нашего движения в верном направлении.

При установке под Windows XP:

- в каталог c:\Program Files\ru\CAD\bin — EXE-файлы и DLL;
- в каталог c:\Documents and Settings\All Users\Application Data\ruCAD\Shared и его подкаталоги (app, block, block-lib, block-lw, dcl, help, html, Layers) — файлы, общие для всех пользователей;
- в каталог c:\Documents and Settings\ShaggyDoc\Local Settings\Application Data\ruCAD \LocalData и его подкаталоги — файлы для настроек конечного пользователя.

Рядом разместились каталоги весьма известных программных продуктов, что также свидетельствует о правильности выбранного пути.

После внесения некоторых изменений в наши программы мы добиваемся, чтобы система ruCAD правильно устанавливалась.

Испытания в работе

На следующем этапе мы начинаем испытывать систему в работе. Сначала мы запускаем ее от имени администратора. Если грубых ошибок нет, то все будет работать правильно.

Далее следует войти в Windows в качестве обычного пользователя. Вот теперь у нас будет очень много проблем. Начнутся они с невозможности записи в файлы. Напомним, что множество настроек записываются в INI-файлы. По сценарию установки мы собрали все файлы в подкаталоги папки All Users. В эту папку могут записывать все пользователи, но файлы копировались от имени администратора. В результате обычный пользователь не сможет внести изменения в обычный INI-файл, потому что владельцем этого файла является администратор. Создать свой файл — можно, а изменить принадлежащий администратору — нельзя.

Если бы мы вообще не копировали настроечные файлы, то все равно столкнулись бы с подобной ситуацией, когда один обычный пользователь не может изменить файл, принадлежащий другому обычному пользователю. Вот мы и "наступили на грабли, которые сами положили на тропинку!" В *главе 3* мы подробно разобрали, где можно хранить настройки, и где мы их будем хранить. Из жалости к пользователям мы решили не слушать рекомендаций Microsoft по хранению настроек в реестре и решили сохранять их в INI-файлах. При разработке сценария инсталляции мы еще более разжалобились и отступили от плана хранения локальных настроек каждого пользователя в его личном каталоге и "свалили" все настроечные файлы в общий каталог — Document and Settings\All Users\Application Data\ruCAD.

Сделано это было в одной строке сценария:

```
Source: cad\LocalData\*;
   DestDir: {commonappdata}\ruCAD\LocalData;
   Components: main; Flags: recursesubdirs
```

Конечно, мы хотели "как лучше" — устанавливая систему от имени администратора сделать файлы доступными всем пользователям, но получилось "как всегда".

Замечание

Мы столкнулись с типичной ситуацией — программы, прекрасно работающие у разработчиков, неработоспособны у конечных пользователей. Увы, так "сработаны" и некоторые популярные программы известных фирм. Мы не будем изображать "хорошую мину при плохой игре", не будем пенять на "злобных админов" и "кривую" Windows. Давайте признаемся, что пока "кривые" только наши руки и мозги, и приступим к корректировке программ.

Корректировка программ

Фактически мы должны переписать примерно четверть книги, но делать этого мы не будем. Мы легко могли бы изобразить, что заранее все знали, и сразу приводить правильные исходные тексты, но наша книга о том, *как это делается*, а не о том, *как это сделано*, а делается многое путем проб и ошибок. Подумаем, что мы можем сделать для обеспечения работоспособности программ при их минимальной переделке.

Прежде всего, отказываемся от пагубной идеи хранить настройки в файлах. Многие настройки, сохранявшиеся в INI-файлах, мы перенесем в реестр. Облегчает работу то, что все операции чтения-записи инкапсулированы в нескольких функциях, и нам необходимо только изменить их реализацию без изменения интерфейсов. Например, нам не нужно заменять в сотнях мест запись в INI-файл последних параметров, достаточно только в одном месте переопределить функцию `ru-user-write-last-param`.

В приложениях, написанных на Delphi, нам достаточно включить у компонентов класса `TJvFormPlacement` свойство `UseRegistry` и задать для свойства `RegistryRoot` значение `HKEY_CURRENT_USER`. Одновременно мы сможем записывать некоторые настройки, ранее передававшиеся в виде аргументов, методами этого компонента (например, номер последнего выбиравшегося элемента в XML-меню).

Любой параметр мы будем читать по схеме: сначала из ветви `HKEY_CURRENT_USER`, если там значение не найдено, то из ветви `HKEY_LOCAL_MACHINE`, если и там не найдено, то принимается значение по умолчанию.

Запись практически всех параметров мы будем делать только в ветвь `HKEY_CURRENT_USER`, и только некоторых — в `HKEY_LOCAL_MACHINE`, да и то в случае, если программа запущена системным администратором.

Новая стратегия работы с пользователями ruCAD

Ранее мы предусматривали работу с пользователями нашей системы при хранении информации о пользователях в *локальной* базе данных. При реальной работе с *такой*

БД разных пользователей Windows NT возникнут проблемы с доступом[1], да и сама идея далека от концепции работы в Windows. Теперь мы убедились, что делать нужно иначе.

Количество пользователей ruCAD на одном компьютере будет не более десятка и обойтись без БД при этом вполне возможно. Всю информацию о пользователях удобнее и надежнее хранить в реестре Windows. Пользователь Windows одновременно становится потенциальным пользователем ruCAD. Администратор ruCAD должен одновременно иметь права члена группы "Администраторы" Windows — иначе он просто не сможет выполнять многие действия по сохранению данных. Перечень прав контролируется более жестко. Например, права редактирования XML или помещения калек в архив могут быть реализованы, только если пользователь вообще имеет право на запись файлов в соответствующие каталоги.

Сведения об *активном пользователе Windows* хранятся в его ветви реестра (а это всегда `HKEY_CURRENT_USER`). Нет необходимости хранить LOGIN и PASSWORD, зато можно добавить дополнительные сведения, например информацию о структурном подразделении.

Если систему ruCAD запускает активный пользователь Windows, ему не потребуется проходить регистрацию. Все его привилегии уже известны. Но под именем одного пользователя Windows в ruCAD могут работать различные пользователи САПР. В Windows 98 это обычная ситуация, в Windows NT это может понадобиться администратору и разработчику для проверки работы системы защиты на уровне ruCAD. Мы предусматриваем возможность ведения списка субпользователей ruCAD.

Право пользователя ruCAD на какое-то действие означает право на выполнение соответствующей функции ruCAD. Реализовать это право функция системы ruCAD сможет только в том случае, если ей это позволит операционная система (фактически — Администратор Windows). Например, в ruCAD предусмотрено право на сохранение калек в электронном архиве. Реализация этого права предусмотрена специальной функцией, которая знает, как и куда положить кальку. Пользователь ruCAD может запустить эту функцию, но она может и не выполнить свою задачу, если Администратор Windows не предоставил пользователю право записи в этот каталог, причем не просто записи, но и замены, возможно, имеющихся в этом каталоге файлов, *владельцами* которых являются иные пользователи. Сам Администратор, имеющий доступ ко всем ресурсам компьютера, может не иметь права на запуск этой функции. Он, конечно, может установить себе любые права и в системе ruCAD, но подумает, прежде чем эту возможность будет использовать — он ведь не знает, какие кальки и куда нужно раскладывать.

В новых условиях можно и отказаться от использования парольного входа в ruCAD. Наши программы, за некоторыми исключениями, не выполняют действий, которые могут нанести вред операционной системе и иным пользователям. Для предотвращения упомянутых исключений требуется работа в защищенной операционной сис-

---

[1] При работе с удаленной клиент-серверной базой данных проблемы доступа решаются не на уровне физического доступа к файлу, а на уровне сервера БД. Информация о пользователях там организована на качественно ином уровне (не в одной примитивной таблице), пользователи могут работать под разными операционными системами и даже через Интернет. Подобным образом, например, организован доступ к электронным картам, размещаемым на Web-сервере в геоинформационной системе.

теме, а там защита осуществляется на уровне политики безопасности Windows NT. В Windows 98 профиль пользователя ruCAD только предоставляет пользователю дополнительные удобства, а система привилегий рассчитана на "законопослушного" человека. "Вредный" пользователь (обычно из категории "ламер") в Windows 98 имеет доступ ко всем ресурсам компьютера и может навредить не только себе, но и соседям по "коммунальному" компьютеру. Надеемся, что в ближайшее время все поймут, что для корпоративной системы такие операционные системы непригодны. Но от паролей мы можем отказаться уже сейчас.

```
[HKEY_CURRENT_USER\Software\ruCAD group\UserData]
TITLE="Достопочтенный"
LONG_NAME="Ипполит Матвеевич"
WORK_GROUP="COMMON"
WORK_DIR=""
DEPARTMENT="Отдел рогов"
```

В этих ключах хранится безопасная информация, которую пользователь может изменять сам, разумеется, с помощью специальной программы, а не прямого редактирования реестра.

Таблица наименований прав и соответствующих им флагов хранится в ветви реестра `HKEY_LOCAL_MACHINE` и заполняется только при инсталляции. Обычный пользователь к этой ветви доступа на запись не имеет.

```
[HKEY_LOCAL_MACHINE\SOFTWARE\ruCAD group\Users\Permissions\Flags]
ADMIN=dword:00000001
...
XML_EDIT=dword:00000800
```

Напомним, что названия ключей, подобные `XML_EDIT`, используются для облегчения нашей собственной работы и фактически являются именованными константами. Для "человеческого" отображения наименований прав используются их описания, хранящиеся в соответствующих ключах.

```
[HKEY_LOCAL_MACHINE\SOFTWARE\ruCAD group\Users\Permissions\Notes]
"ADMIN"="Администрирование"
...
"XML_EDIT"="Редактирование XML"
```

Требования к Администратору Windows при установке ruCAD повышаются. Он должен представлять, что и как делает система и какие могут быть последствия этих действий. Разумный Администратор Windows предоставит соответствующие привилегии пользователям ruCAD. Он не будет сам бегать по всем рабочим местам, на которых пользователь решит изменить, например, иллюстрацию в XML-меню, а предоставит доступ на запись в этот каталог. Возможно, он перенесет каталог с иллюстрациями в другое место или даже на другой компьютер — для этого Администратор должен знать, что и куда можно переносить. Наша задача — обеспечить его подробной документацией и предоставить соответствующие инструменты.

Замечание

А что произойдет, если все-таки права на запись какого-то файла нет? Произойдет ошибка, но все такие ошибки разработчики обязаны предвидеть и предусматривать соответствующие решения — внятные сообщения и тому подобные действия, этим вопросам мы

отдали много места в книге. Пока мы работали с программами с правами Администратора, сообщений об ошибках не было. Как только мы запустили программу-стартер под профилем обычного Пользователя — появились сообщения, послужившие причиной переработки стратегии работы с файлами, но программа продолжала работу. Если бы мы не предусмотрели обработки исключительных ситуаций, то до сих пор гадали бы, почему "падают" программы.

Устанавливает права Администратор Windows с помощью специальной утилиты. Записываются права в "машинную" ветвь реестра, где по идентификатору пользователя задается сумма битов, соответствующая набору прав пользователя.

Самые большие изменения придется вносить в программу-стартер. Мы исключим из нее процедуру регистрации пользователя при входе, но введем специальный модуль **Менеджер безопасности**, позволяющий настроить систему. Измененные исходные тексты мы не приводим, т. к. потребуется слишком много места на код "не про AutoCAD". Кроме того, мы не рискуем публиковать деликатные коды управления системой безопасности Windows — предоставим это право более компетентным специалистам. Приведем только несколько иллюстраций готовых решений. На рис. 36.7 показана закладка редактирования свойств пользователя. Безопасную личную информацию может редактировать сам пользователь, а изменять привилегии, добавлять и удалять пользователей — только Администратор Windows.

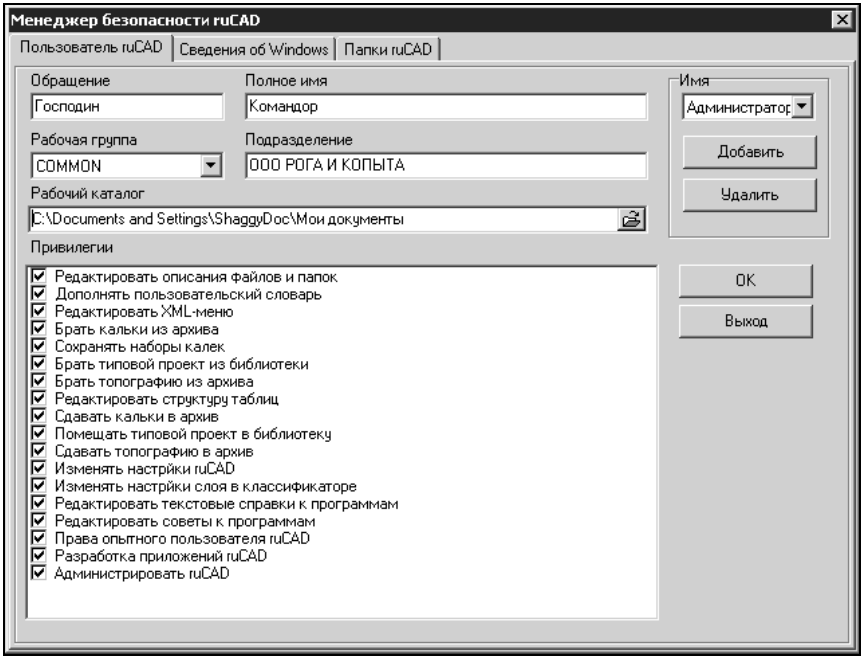


Рис. 36.7. Редактор свойств пользователя

На рис. 36.8 показано отображение свойств операционной системы. Более подробная информация доступна при щелчке по кнопке **Подробнее**, запускающей системную программу msinfo.

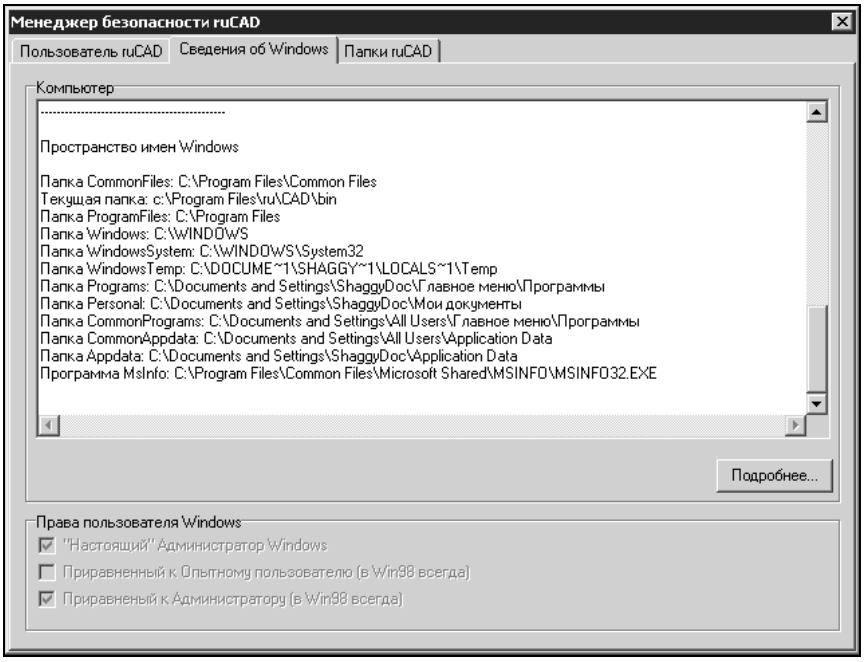


Рис. 36.8. Просмотр сведений о Windows

На вкладке **Папки ruCAD** (рис. 36.9) Администратор Windows сможет просмотреть расположение именованных папок ruCAD, перенести их при необходимости в другие каталоги (будут перенесены и файлы и изменены ключи реестра), просмотреть и изменить привилегии для групп пользователей. Все эти опасные функции выполняются стандартными средствами операционной системы.

Теперь программа-стартер знает об операционной системе и пользователе все, что необходимо. Делаем следующий шаг — передаем все эти сведения нашим LISP-функциям. Конечно, из них мы можем просто прочитать реестр, но вряд ли целесообразно постоянно "копаться" в реестре. Лучше мы передадим все сведения в виде одной глобальной переменной[1], для чего в очередной раз изменим процедуру генерирования файла acaddoc.lsp.

Этот файл по-прежнему генерируется из шаблона, но его первая часть создается стартером путем записи всей известной ему информации. При чтении готового файла acaddoc.lsp (листинг 36.14) будет создана глобальная переменная `*ru_startup_info*`, представляющая собой ассоциированный список. К элементам этого списка можно легко добираться по их именам.

[1] Нельзя забывать о поддержании актуальности информации, хранящейся в этой глобальной переменной. Если по ходу выполнения какой-то функции необходимо изменить значение, хранящееся в реестре, то обязательно должна быть предусмотрена синхронизация содержимого реестра и глобальной переменной.

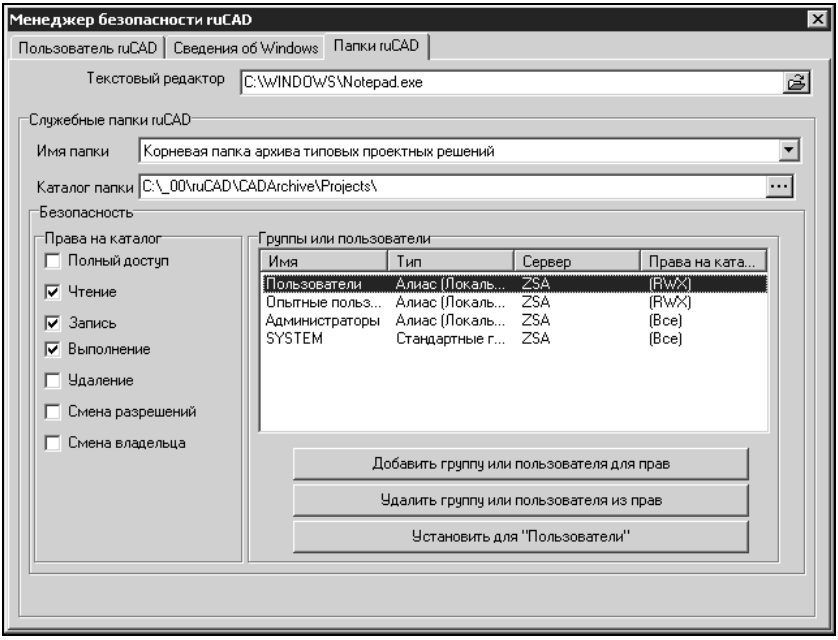


Рис. 36.9. Настройка прав доступа к каталогам

Листинг 36.14. Окончательный вариант файла acaddoc.lsp

```
(setq *ru_startup_info*
 (list
  (cons "FoldersInfo"
   (list
    (cons "FoldersNotes"
     (list
      (cons "RootDir" "Корневой каталог ruCAD")
      (cons "AppDataDir" "Общие данные сторонних приложений")
      (cons "LocalSettingsDir" "Локальные настройки и данные")
      (cons "LocalAppDataDir" "Локальные данные приложений ruCAD")
      (cons "LocalAcadAllVersionDir"
         "Папка файлов поддержки AutoCAD, общих для всех версий")
      (cons "AllUsersDir"
         "Корневая папка общих данных всех пользователей")
      (cons "CurrentUserDir"
         "Корневая папка общих данных текущего пользователя")
      (cons "LspSourceDir" "Исходные тексты LISP-функций")
      (cons "HtmlDir" "Корневая папка HTML-файлов")
      (cons "XmlMenuDir" "Корневая папка XML-меню")
      (cons "XmlImagesDir" "Корневая папка иллюстраций к XML-меню")
      (cons "LayersClassDir" "Корневая папка классификатора слоев")
      (cons "ArchiveTopoDir" "Корневая папка архива топографии")
```

```
      (cons "ArchiveProjectsDir"
              "Корневая папка архива типовых проектных решений")
      (cons "ArchiveLayersDir" "Корневая папка архива калек-слоев")
    )
   )
   (cons "FoldersDirs"
    (list
      (cons "RootDir" "C:\\Program Files\\ru\\CAD\\")
      (cons "AppDataDir"
        "C:\\Documents and Settings\\All Users\\Application
Data\\ru\\CAD\\Shared\\")
      (cons "LocalSettingsDir"
          "C:\\Documents and Settings\\All Users\\Application
Data\\ru\\CAD\\LocalData\\")
      (cons "LocalAppDataDir" "C:\\Documents and Settings\\All Users\\Application
Data\\ru\\CAD\\LocalData\\")
      (cons "LocalAcadAllVersionDir" "C:\\Documents and Settings\\All
Users\\Application Data\\ru\\CAD\\LocalData\\AutoCAD\\")
      (cons "AllUsersDir" "C:\\Documents and Settings\\All Users\\Application
Data\\ru\\CAD\\Shared\\")
      (cons "CurrentUserDir" "C:\\Documents and Settings\\All Users\\Application
Data\\ru\\CAD\\UsersData\\")
      (cons "LspSourceDir" "C:\\.ru\\_ruSource\\lisp\\")
     (cons "HtmlDir" "C:\\Documents and Settings\\All Users\\Application Data\\
ru\\CAD\\Shared\\Html\\")
      (cons "XmlMenuDir" "C:\\Documents and Settings\\All Users\\Application Data
\\ru\\CAD\\Shared\\Xml\\Menu\\")
      (cons "XmlImagesDir" "C:\\Documents and Settings\\All Users\\Application Data
\\ru\\CAD\\Shared\\Xml\\Images\\")
      (cons "LayersClassDir" "C:\\Documents and Settings\\All Users\\Application
Data\\ru\\CAD\\Shared\\Layers\\Все\\")
      (cons "ArchiveTopoDir" "C:\\Documents and Settings\\All Users\\Application
Data\\ru\\CADarchive\\CADArchive\\Topo\\")
      (cons "ArchiveProjectsDir" "C:\\Documents and Settings\\All
Users\\Application Data\\ru\\CADarchive\\CADArchive\\Projects\\")
      (cons "ArchiveLayersDir" "C:\\Documents and Settings\\All Users\\Application
Data\\ru\\CADarchive\\CADArchive\\Layers\\")
    )
   )
 )
)
 (cons "UserInfo"
  (list
   (cons "LOGIN" "Администратор")
   (cons "TITLE" "Господин")
   (cons "LONG_NAME" "Командор")
   (cons "WORK_GROUP" "COMMON")
   (cons "WORK_DIR" "C:\\.ru\\cad\\samples\\dwg")
   (cons "DEPARTMENT" "ООО РОГА И КОПЫТА")
   (cons "BIT_RIGHTS" 262143)
   (cons "RU_IS_CAN_REG_EDIT" T)
 )
)
```

```
 (cons "PermissionsInfo"
  (list
  (cons "PermissionsNotes"
   (list
    (cons "EDIT_DIRINFO" "Редактировать описания файлов и папок")
    (cons "ADD_USER_DIC" "Дополнять пользовательский словарь")
    (cons "XML_EDIT" "Редактировать XML-меню")
    (cons "GET_LAY" "Брать кальки из архива")
    (cons "SAVE_LAYERS_SET" "Сохранять наборы калек")
    (cons "GET_PROJ" "Брать типовой проект из библиотеки")
    (cons "GET_TOPO" "Брать топографию из архива")
    (cons "TABLE_STRU_EDIT" "Редактировать структуру таблиц")
    (cons "PUT_LAY" "Сдавать кальки в архив")
    (cons "PUT_PROJ" "Помещать типовой проект в библиотеку")
    (cons "PUT_TOPO" "Сдавать топографию в архив")
    (cons "SETUP_CAD" "Изменять настройки ruCAD")
    (cons "SETUP_LAY" "Изменять настройки слоя в классификаторе")
    (cons "TXT_HLP_EDIT" "Редактировать текстовые справки к программам")
    (cons "TIPS_EDIT" "Редактировать советы к программам")
    (cons "PROFI" "Права опытного пользователя ruCAD")
    (cons "DEVELOP" "Разработка приложений ruCAD")
    (cons "ADMIN" "Администрировать ruCAD")
   )
  )
  (cons "PermissionsFlags"
   (list
    (cons "EDIT_DIRINFO" 1)
    (cons "ADD_USER_DIC" 2)
    (cons "XML_EDIT" 4)
    (cons "GET_LAY" 8)
    (cons "SAVE_LAYERS_SET" 16)
    (cons "GET_PROJ" 32)
    (cons "GET_TOPO" 64)
    (cons "TABLE_STRU_EDIT" 128)
    (cons "PUT_LAY" 256)
    (cons "PUT_PROJ" 512)
    (cons "PUT_TOPO" 1024)
    (cons "SETUP_CAD" 2048)
    (cons "SETUP_LAY" 4096)
    (cons "TXT_HLP_EDIT" 8192)
    (cons "TIPS_EDIT" 16384)
    (cons "PROFI" 32768)
    (cons "DEVELOP" 65536)
    (cons "ADMIN" 131072)
   )
  )
 )
 )
(cons "WinInfo" (list (cons "IsWinNT" T)))))
(vl-load-com)
(setq *ru_root_dir* "C:\\Program Files\\ru\\CAD\\")
```

```
(load "C:\\Documents and Settings\\All Users\\Application Data\\ru\\CAD\\Shared
\\app\\ru-lib-main.fas")
(if (< (atoi "15") 16)
 (progn
  (if (not (member "ru_MainLib.arx" (arx)))
    (arxload (findfile "ru_MainLib.arx")))
    (setq *ru_use_ru_arx* T)
);_ end of progn
  (setq *ru_use_ru_arx* nil)
)
(ru-express-load)
(ru-doslib-load)
(if (or (ru-doslib-load) *ru_use_ru_arx*)
 (ru-init-start-rucad)
 (alert "Система не сможет работать. Не загружена ни MainLib, ни DosLib")
)
```

После загрузки этого файла мы знаем всю необходимую информацию.

Приведем примеры использования информации о пользователе. В листинге 36.15 показано, как из глобальной переменной извлекается информация о пользователе.

Листинг 36.15. Функция ru-user-get-current-user-info

```
(defun ru-user-get-current-user-info ()
    (cdr (assoc "UserInfo" *ru_startup_info*))
);_ end of defun
```

Эта функция вернет список

```
 (("LOGIN" . "Администратор") ("TITLE" . "Господин")
  ("LONG_NAME" . "Командор")("WORK_GROUP" . "COMMON")
  ("WORK_DIR" . "C:\\.ru\\cad\\samples\\dwg")
  ("DEPARTMENT" . "ООО РОГА И КОПЫТА")
  ("BIT_RIGHTS" . 262143)("RU_IS_CAN_REG_EDIT" . t)
)
```

Все сведения о привилегиях можно получить так, как показано в листинге 36.16.

Листинг 36.16. Функция ru-user-get-permissions-info

```
(defun ru-user-get-permissions-info ()
  (cdr (assoc "PermissionsInfo" *ru_startup_info*))
);_ end of defun
```

Функция `ru-user-get-permissions-flags` (листинг 36.17) вернет ассоциированный список имен и флагов привилегий пользователя.

Листинг 36.17. Функция ru-user-get-permissions-flags

```
(defun ru-user-get-permissions-flags ()
  (cdr (assoc "PermissionsFlags" (ru-user-get-permissions-info)))
);_ end of defun
```

Функция `ru-user-get-permissions-flags-list` (листинг 36.18) вернет список флагов привилегий пользователя.

Листинг 36.18. Функция `ru-user-get-permissions-flags-list`

```
(defun ru-user-get-permissions-flags-list ()
  (mapcar 'cdr (ru-user-get-permissions-flags))
);_ end of defun
```

Функция `ru-user-get-permissions-names-list` (листинг 36.19) вернет список имен привилегий пользователя.

Листинг 36.19. Функция `ru-user-get-permissions-names-list`

```
(defun ru-user-get-permissions-names-list ()
  (mapcar 'car (ru-user-get-permissions-notes))
);_ end of defun
```

Функция `ru-user-get-permissions-notes` (листинг 36.20) вернет ассоциированный список имен и описаний привилегий пользователя.

Листинг 36.20. Функция `ru-user-get-permissions-notes`

```
(defun ru-user-get-permissions-notes ()
  (cdr (assoc "PermissionsNotes" (ru-user-get-permissions-info)))
);_ end of defun
```

Функция `ru-user-get-permissions-notes-list` (листинг 36.21) вернет список описаний привилегий пользователя.

Листинг 36.21. Функция `ru-user-get-permissions-notes-list`

```
(defun ru-user-get-permissions-notes-list ()
  (mapcar 'cdr (ru-user-get-permissions-notes))
);_ end of defun
```

Семейство функций, подобных `ru-user-title` (листинги 36.22 и 36.23), возвращают значения свойств пользователя (в примерах — `TITLE` и `LONG_NAME`).

Листинг 36.22. Функция `ru-user-title`

```
(defun ru-user-title ()
 (cdr (assoc "TITLE" (ru-user-get-current-user-info)))
)
```

Листинг 36.23. Функция `ru-user-long-name`

```
(defun ru-user-long-name ()
 (strcat  (ru-user-title) " "
```

```
 (cdr (assoc "LONG_NAME" (ru-user-get-current-user-info))))
);_ end of defun
```

Функция `ru-user-right-flag` (листинг 36.24) вернет целое число, определяющее набор привилегий пользователя.

Листинг 36.24. Функция `ru-user-right-flag`

```
(defun ru-user-right-flag ()
  (cdr (assoc "BIT_RIGHTS" (ru-user-get-current-user-info)))
);_ end of defun
```

Функция `ru-user-perm-bit-by-name` (листинг 36.25) вернет целое число, соответствующее флагу именованной привилегии.

Листинг 36.25. Функция `ru-user-perm-bit-by-name`

```
(defun ru-user-perm-bit-by-name (name)
  (cond
    ((cdr (assoc name (ru-user-get-permissions-flags))))
    (t -1)
 );_ end of cond
);_ end of defun
```

Функция, приведенная в листинге 36.26, проверяет разрешения пользователя на выполнение именованного действия.

Листинг 36.26. Функция `ru-user-test-permission`

```
(defun ru-user-test-permission (name)
  (ru-match-is-bit-in-flag
    (ru-user-perm-bit-by-name name)
    (ru-user-right-flag)
 );_ end of ru-match-is-bit-in-flag
);_ end of defun
```

Семейство функций, одна из которых приведена в листинге 36.27, проверяют разрешения пользователя на выполнение конкретных именованных действий.

Листинг 36.27. Функция `ru-user-may-xml-edit`

```
(defun ru-user-may-xml-edit ()
  (ru-user-test-permission "XML_EDIT")
);_ end of defun
```

Функция, показанная в листинге 36.28, выводит диалоговое окно со списком наименований привилегий и отметками об их доступности для конкретного пользователя без возможности редактирования.

Листинг 36.28. Функция ru-user-show-rights

```
(defun ru-user-show-rights ()
  (ru-dlg-show-check-list
    (strcat (ru-user-long-name) " - ваши права:")
    (ru-user-get-permissions-notes-list)
    (mapcar 'ru-user-test-permission
     (ru-user-get-permissions-names-list))
    nil
 );_ end of ru-dlg-user-show-check-list
  (princ)
);_ end of defun
```

В конечных программах используется функция, приведенная в листинге 36.29. Она применяется во всех потенциально опасных программах, и извещает пользователя об отсутствии у него прав на какое-либо действие. Если аргумент `show_msg` равен `nil`, то сообщение не выводится.

Листинг 36.29. Функция ru-user-action-enabled

```
(defun ru-user-action-enabled (name_right show_msg)
;;; Делает остановку с сообщением о нарушении права
;;; Если все нормально, пропускает дальше
;;; Флаг show_msg для возможности тихой проверки
;;;(ru-user-action-enabled "ADMIN" T)
;;;(ru-user-action-enabled "GET_TOPO" nil)
;;;(ru-user-action-enabled "ПРЫГАТЬ" T)
  (cond ((ru-user-test-permission name_right) T)
   (T
    (if show_msg
      (ru-msg-alert
       (strcat
         "\nВы не имеете разрешения на\n"
         (ru-user-perm-note-by-name name_right)
    ));_ end of ru-msg-alert
   );_ end of if
 nil
))
);_ end of defun
```

Примерно так же обрабатываются и другие подсписки глобальной переменной `*ru_startup_info*`.

Корректировка сценария установки

С учетом всех изменений откорректируем сценарий установки. Прежде всего доработаем секцию `Code` (листинг 36.30), дополнив ее выбором папок для файлов общего доступа и архивов.

Листинг 36.30. Файл сценария ruCADBookLT.iss. Измененная секция Code

```
[Code]
var
 ArchiveDir: String;
 SharedDir: String;
 TxtEditor: String;

// Выбор AutoCAD в диалоге
function SelectAcad():boolean;
external 'SelectAcad@files:ruSelAcad.dll stdcall';

function ruKey(Default: String): String;
begin
  Result := 'SOFTWARE\ruCAD group\ruCAD';
end;

function PromptArchiveDir(BackClicked: Boolean): Boolean;
// Выбор каталога для архивов
var
  Next: Boolean;
  CommonDefDir: string;
begin
    CommonDefDir:=ExpandConstant('{commonappdata}');
    // Открываем дополнительную страницу мастера
    ScriptDlgPageOpen();
    ScriptDlgPageSetCaption('Папка для АРХИВОВ');
    ScriptDlgPageSetSubCaption1('Архивы, калек-слоев, типовых проектных решений
и топографии');
    ScriptDlgPageSetSubCaption2('К файлам этой папки должен быть установлен доступ
для чтения!');
    ArchiveDir := CommonDefDir + '\ru\CADarchive';
    Next := InputDir( 'ru\CADArchive', ArchiveDir);
    while Next and (ArchiveDir = '') do begin
      MsgBox(SetupMessage(msgInvalidPath), mbError, MB_OK);
      Next := InputDir('ru\CADArchive', ArchiveDir);
    end;
    if not BackClicked then Result := Next else Result := not Next;
    ScriptDlgPageClose(not Result);
end;

// Выбор каталога для общих файлов
function PromptSharedDir(BackClicked: Boolean): Boolean;
var
  Next: Boolean;
  CommonDefDir: string;
begin
    CommonDefDir:= ExpandConstant('{commonappdata}');
    ScriptDlgPageOpen();
    ScriptDlgPageSetCaption('Папка для файлов общего доступа - WinNT');
```

```
    ScriptDlgPageSetSubCaption1('Компоненты, используемые всеми пользователями');
    ScriptDlgPageSetSubCaption2('Ко всем файлам этой папки должен быть установлен
полный доступ!');
    SharedDir := CommonDefDir + '\ru\CAD';
    Next := InputDir( 'ru\CAD', SharedDir);
    while Next and (SharedDir = '') do begin
      MsgBox(SetupMessage(msgInvalidPath), mbError, MB_OK);
      Next := InputDir('ru\CAD', SharedDir);
    end;
    if not BackClicked then Result := Next else Result := not Next;
    ScriptDlgPageClose(not Result);
end;
// Выбор текстового редактора
function PromptTxtEditor(BackClicked: Boolean): Boolean;
var
  Next: Boolean;
begin
    ScriptDlgPageOpen();
    ScriptDlgPageSetCaption('Выбор текстового редактора');
    ScriptDlgPageSetSubCaption1('Текстовый редактор, используемый для правки
пользовательских текстов');
    ScriptDlgPageSetSubCaption2('Пользователь должен иметь право запуска этой
программы! Выберите');
    TxtEditor := ExpandConstant('{win}') + '\Notepad.exe';
    Next := InputFile('Выбор текстового редактора', 'Приложения (*.exe)|*.exe',
'.exe', TxtEditor);
    while Next and (TxtEditor = '') do begin
      MsgBox(SetupMessage(msgInvalidPath), mbError, MB_OK);
      Next := InputFile('Выбор текстового редактора', 'Приложения (*.exe)|*.exe',
'.exe', TxtEditor);
    end;
    if not BackClicked then Result := Next else Result := not Next;
    ScriptDlgPageClose(not Result);
end;

function ScriptDlgPages(CurPage: Integer; BackClicked: Boolean): Boolean;
begin
  if (not BackClicked and (CurPage = wpSelectDir)) or (BackClicked and (CurPage =
wpReady)) then begin
    Result := PromptSharedDir(BackClicked);
    Result := PromptArchiveDir(BackClicked);
  end else Result := True;
end;

function ScriptTxtEditor(CurPage: Integer; BackClicked: Boolean): Boolean;
begin
  if (not BackClicked and (CurPage = wpSelectTasks)) or (BackClicked and (CurPage =
wpReady))
   then Result := PromptTxtEditor(BackClicked) else Result := True;
end;
```

```
function GetSharedDir(S: String): String;
begin
  Result := SharedDir;
end;

function GetArchiveDir(S: String): String;
begin
  Result := ArchiveDir;
end;

function GetTxtEditor(S: String): String;
begin
  Result := TxtEditor;
end;

function UpdateReadyMemo(Space, NewLine, MemoUserInfoInfo, MemoDirInfo,
MemoTypeInfo,
 MemoComponentsInfo, MemoGroupInfo, MemoTasksInfo: String): String;
var
  S: String;
begin
  S := MemoDirInfo + NewLine + NewLine;
  S := S + 'Папка для общих файлов ' + NewLine;
  S := S + Space + SharedDir + NewLine;
  S := S + 'Папка для архивов ' + NewLine;
  S := S + Space + ArchiveDir + NewLine;
  S := S + 'Текстовый редактор ' + NewLine;
  S := S + Space + TxtEditor + NewLine;
  Result := S;
end;

function BackButtonClick(CurPage: Integer): Boolean;
begin
  Result := (ScriptDlgPages(CurPage, True) and ScriptTxtEditor(CurPage, True));
end;

function NextButtonClick(CurPage: Integer): Boolean;
begin
  Result := True;
  case CurPage of
    wpInfoBefore : begin
       Result:= SelectAcad;
      if not result then MsgBox('Не выбрана рабочая система AutoCAD!'+#13
       +'Отмените установку или повторите выбор', mbCriticalError,MB_OK);
   end;
    wpSelectDir: ScriptDlgPages(CurPage, False);
    wpSelectTasks: ScriptTxtEditor(CurPage, False);
  end;
end;
```

Теперь мы можем использовать переменные, вычисленные в секции `Code` и в других секциях сценария (листинг 36.31).

Листинг 36.31. Примеры изменений в секции `Files`

```
[Files]
;-------------------------------------------------------------------
;; Используем результат функции GetSharedDir
;; из секции Code в качестве константы{code:GetSharedDir|''}
;-------------------------------------------------------------------
Source: cad\Shared\*;
       DestDir: {code:GetSharedDir|''}\Shared;
       Components: main; Flags: recursesubdirs
;-------------------------------------------------------------------
;; Используем результат функции GetArchiveDir
;; из секции Code в качестве константы{code:GetArchiveDir|''}
;-------------------------------------------------------------------
Source: cad\Archive\*; DestDir:
       {code:GetArchiveDir|''}\CADArchive;
       Components: main add_arc;
       Flags: recursesubdirs; Tasks: arc_layers arc_projects arc_topo
;-------------------------------------------------------------------
; Также отправляем в более ближние каталоги LocalData и UsersData
; Но для этого, чтобы не описывать каждый подкаталог,
; просто перенесем их выше посредством сценария
;-------------------------------------------------------------------
Source: cad\LocalData\*;
        DestDir: {code:GetSharedDir|''}\LocalData;
        Components: main; Flags: recursesubdirs
;-------------------------------------------------------------------
Source: cad\UsersData\*;
        DestDir: {code:GetSharedDir|''}\UsersData;
        Components: main; Flags: recursesubdirs
;-------------------------------------------------------------------
```

Внесем в секцию `Tasks` дополнительные задачи по установке архивов калек, типовых проектных решений и топографии (листинг 36.32).

Листинг 36.32. Изменения в секции `Tasks`

```
[Tasks]
Name: arc_topo; Description: Архив топографических планшетов; GroupDescription:
Архивные материалы; Components: add_arc
Name: arc_projects; Description: Типовые проектные решения; GroupDescription:
Архивные материалы; Components: add_arc
Name: arc_layers; Description: Кальки-слои; GroupDescription: Архивные материалы;
Components: add_arc
```

Внесем изменения в секцию `Registry` (листинг 36.33). В ней мы также будем использовать в виде именованных констант результаты функций из секции `Code`, но

главное, что записей в реестр будет намного больше (мы приводим только фрагменты), т. к. именно программа установки должна создать все требуемые настройки. В наших программах больше не "зашиты" не только расположение каталогов, но и их "человеческие" описания.

Листинг 36.33. Фрагменты секции `Registry`

```
[Registry]
;---------- Каталоги -------------------------------------------------
Root: HKLM; SubKey: {code:ruKey}; ValueType: string;
    ValueName: RootDir; ValueData: {app};
    Flags: uninsdeletekey; Components: main
Root: HKLM; SubKey: {code:ruKey}; ValueType: string;
    ValueName: AppDataDir; ValueData: {code:GetSharedDir|''}\Shared;
    Flags: uninsdeletekey; Components: main
...
Root: HKLM; SubKey: {code:ruKey}; ValueType: string;
     ValueName: ArchiveTopoDir;
     ValueData: {code:GetArchiveDir|''}\CADArchive\Topo;
     Flags: uninsdeletekey; Components: main add_arc; Tasks: arc_topo
...
;---------- Описания каталогов----------------------------------------
Root: HKLM; SubKey: {code:ruKey}\Folders\Notes; ValueType: string;
     ValueName: RootDir; ValueData: Корневой каталог ruCAD;
     Flags: uninsdeletekey; Components: main
Root: HKLM; SubKey: {code:ruKey}\Folders\Notes; ValueType: string;
     ValueName: AppDataDir; ValueData: Общие данные сторонних приложений;
     Flags: uninsdeletekey; Components: main
...
Root: HKLM; SubKey: {code:ruKey}\Folders\Notes; ValueType: string;
     ValueName: ArchiveTopoDir;
     ValueData: Корневая папка архива топографии;
     Flags: uninsdeletekey; Components: main add_arc; Tasks: arc_topo
...
;---------- Имена и значения привилегий------------------------------
Root: HKLM; SubKey: {code:ruKey}\Users\Permissions\Flags;
      ValueType: dword; ValueName: EDIT_DIRINFO; ValueData: 1;
      Flags: uninsdeletekey; Components: main
...
Root: HKLM; SubKey: {code:ruKey}\Users\Permissions\Flags;
     ValueType: dword; ValueName: ADMIN; ValueData: 131072;
     Flags: uninsdeletekey; Components: main
;---------- Описания привилегий--------------------------------------
Root: HKLM; SubKey: {code:ruKey}\Users\Permissions\Notes;
     ValueType: string; ValueName: EDIT_DIRINFO;
     ValueData: Редактировать описания файлов и папок;
     Flags: uninsdeletekey; Components: main
...
Root: HKLM; SubKey: {code:ruKey}\Users\Permissions\Notes;
     ValueType: string; ValueName: ADMIN;
```

```
     ValueData: Администрировать ruCAD;
     Flags: uninsdeletekey; Components: main
;----------- Перечень типовых пользователей и их привилегий------------
Root: HKLM; SubKey: {code:ruKey}\Users; ValueType: dword;
     ValueName: Администратор; ValueData: 262143;
      Flags: uninsdeletekey; Components: main
...
;---------- Личные  данные одного пользователя -------------------------
Root: HKLM; SubKey: {code:ruKey}\Users\Администратор;
     ValueType: string; ValueName: LONG_NAME;
     ValueData: Командор; Flags: uninsdeletekey; Components: main
Root: HKLM; SubKey: {code:ruKey}\Users\Администратор;
     ValueType: string; ValueName: WORK_DIR; ValueData:;
     Flags: uninsdeletekey; Components: main
Root: HKLM; SubKey: {code:ruKey}\Users\Администратор; ValueType: string;
     ValueName: TITLE; ValueData: Господин;
     Flags: uninsdeletekey; Components: main
Root: HKLM; SubKey: {code:ruKey}\Users\Администратор; ValueType: string;
     ValueName: WORK_GROUP; ValueData: COMMON;
     Flags: uninsdeletekey; Components: main
Root: HKLM; SubKey: {code:ruKey}\Users\Администратор; ValueType: string;
     ValueName: DEPARTMENT; ValueData: ООО РОГА И КОПЫТА;
     Flags: uninsdeletekey; Components: main
; ------- Данные по умолчанию для новых пользователей ---------------
Root: HKU; SubKey: .DEFAULT\{code:ruKey}\Users; ValueType: dword;
     ValueName: Новичок; ValueData: 3;
     Flags: uninsdeletekey; Components: main
Root: HKU; SubKey: .DEFAULT\{code:ruKey}\Users\Новичок;
     ValueType: string; ValueName: TITLE; ValueData: Уважаемый;
      Flags: uninsdeletekey; Components: main
Root: HKU; SubKey: .DEFAULT\{code:ruKey}\Users\Новичок;
      ValueType: string; ValueName: LONG_NAME;
      ValueData: Пользователь ruCAD;
      Flags: uninsdeletekey; Components: main
```

Полученный сценарий еще далек от совершенства. Нет в нем, например, определения расположения уже установленной системы ruCAD, нет вариантов простого обновления настроек. Пока не решен и главный вопрос — установка разрешений на папки для любых пользователей после инсталляции системы Администратором Windows. Программа Inno Setup, к сожалению, не позволяет этого сделать. Решить такой вопрос можно путем автоматического запуска консольной программы **SetAcl** (есть несколько программ с таким названием) и установки с ее помощью разрешений. Наиболее известная из программ такого класса имеет огромное количество ключей и параметров, позволяющих сделать практически все. Пока мы ее применять не будем, т. к. есть обоснованные подозрения, что делает она все, но не все правильно — как всегда проблемы с длинными и русскими именами. Придется сначала поэкспериментировать на себе.

Мы уже предусмотрели альтернативное решение — установку разрешений из нашего менеджера безопасности. Воспользуется ли этим предложением Администратор —

его дело. Для хитрого Администратора у нас предусмотрено и недокументированное решение — размещение папок общего доступа вне пространства стандартных имен Windows, т. е. в обычный каталог, например, c:\ru\CAD. Администратор должен догадаться, что такой вновь созданный каталог не наследует стандартных разрешений.

Хорошая новость!

Буквально на следующий день после того, как были написаны эти строки, появилась новая версия программы Inno Setup 4.1.1, в которой предусмотрена долгожданная возможность установки разрешений на каталоги и файлы в процессе инсталляции.

Теперь достаточно указать в сценарии флаг `Permissions`, записываемый в виде

`<идентификатор пользователя или группы>-<тип доступа>`,

где `тип доступа` может иметь одно из предопределенных значений `modify` (чтение и запись), `readexec` (чтение и выполнение) или `full` (полный доступ).

Например, строка сценария

```
Source: cad\Archive\*; DestDir: {code:GetArchiveDir|''}\CADArchive;
                  Components: main add_arc; Flags: recursesubdirs;
                  Tasks: arc_layers arc_projects arc_topo;
                  Permissions: "authusers-modify"
```

предоставит всем зарегистрированным пользователям права на чтение и запись в каталог архива.

Разумеется, мы немедленно внесем коррективы в сценарий установки нашей системы.

Резюме

Разобрав технологию создания установочного комплекта, мы убедились, что даже на самом последнем этапе разработки программы можно обнаружить множество ошибок, но теперь точно знаем, как их можно избежать.

Мы создавали установочный комплект для крупной системы, но аналогичные проблемы могут возникнуть и при разработке единственного приложения для AutoCAD, особенно если оно занимается записью информации.

Полученный установочный комплект надо еще много тестировать в различных условиях. Даже "невинная" Windows 98 по-разному ведет себя при отсутствии профилей пользователей, при их наличии, при входе под профилем пользователя, но с отменой ввода пароля. А есть еще и Windows ME, и различные клоны NT.

Кажущийся последним шаг на пути превращения программы в продукт может оказаться возвратом в самое начало — об этом надо помнить.

ГЛАВА 37

Подготовка к распространению

В этой главе, в отличие от предыдущих, мы будем говорить не о *нашей* системе, а о *вашей*. Предположим, что вы, наши читатели, с нашей скромной помощью разработали собственную САПР (некую SuperCAD), причем намного лучше описываемой в этой книге. Рано или поздно, но работа по созданию *вашей* системы завершена. Вы знаете, что имеется еще много недоделок и ошибок, но в целом система уже работает, а некоторые члены коллектива разработчиков начинают произносить любимую фразу Михаила Самюэлевича Паниковского: "Отдайте мне мои деньги!".

Вы начитались легенд о невероятных доходах авторов популярных программ и начинаете подумывать о том, что можете теперь и неплохо заработать на продаже своей суперпрограммы. А теперь вспомните, что сказал незабвенный товарищ Бендер по аналогичному поводу: "Только вы, дорогой товарищ, плюньте на все это. Слюной, как плевали до эпохи исторического материализма. Ничего не выйдет".

Оставьте в покое свою "голубую мечту" и попробуйте подойти к вопросу распространения (заметьте, еще не продаж) с реалистичных позиций.

Знатоки бизнес-модели условно-бесплатного распространения программ считают, что приложения к популярным базовым пакетам, таким как наша любимая система AutoCAD, имеют большие шансы на успех — пристроившись за мировым лидером можно вырваться вперед. Увы, это не совсем верно.

Особенности приложений для AutoCAD

В общем случае действительно заработок на приложениях к системе AutoCAD реальнее, чем попытка разработать (это возможно[1]) и продвинуть на рынок гипотетический самостоятельный "русский CAD", но фактически эта ниша давно занята. Успехом могут пользоваться отдельные приложения, решающие частные задачи вне AutoCAD (просмотрщики, конверторы), и приложения, облегчающие работу пользователя внутри AutoCAD.

Программы первого типа необходимы далеко не всем пользователям, да и практически все, что требуется, уже давно разработано. Отдельные успехи могут быть и на

[1] Примером является очень хорошая система Компас-График, хотя и не имеющая таких возможностей, как у AutoCAD, но уверенно занимающая свою нишу на российском рынке.

этом направлении. Примером такой программы может служить **ABViewer**[1], разработанная российскими программистами. Программа умеет просматривать множество графических форматов (ну, этим не удивишь), включая DWG и DXF — а это уже интересно. Кроме того, имеется ряд полезных опций, в том числе возможность сохранения выбираемых фрагментов в растровый формат. Быстрый, даже на медленных машинах, просмотр DWG-файлов весьма непростая задача, особенно с учетом того, что формат DWG является закрытым. Хотя программа ABViewer и нуждается в некоторых усовершенствованиях, она вполне может распространяться во всем мире в качестве классической условно-бесплатной программы.

> Здесь мы пишем, что ABViewer может распространяться как Shareware, но на одном из форумов я писал, что "программа на Shareware пока не тянет". Такой преждевременный вывод я сделал потому, что программа в пробном режиме через несколько секунд напоминала о необходимости регистрации. Оценить ее реальные возможности в таких условиях было трудно. Только после получения лицензии достоинства программы были оценены в полной мере. Излишняя назойливость ("дэнги, дэнги давай!"), на мой взгляд, вредит хорошей программе. *(Сергей Зуев)*

Программы второго типа, работающие внутри системы AutoCAD, распространять сложнее. Благодаря развитым средствам разработки в мире разработаны тысячи программ такого типа, большинство из которых решает отдельные задачи или группы сходных задач. О том, что российские программисты, а еще чаще инженеры, с успехом занимаются разработками подобных программ, мы писали во введении, да и наша книга написана в помощь таким разработчикам. Как правило, такие программы в России распространяются бесплатно. Отдельные интересные программы вполне пригодны для условно-бесплатного распространения. Примером может быть программа **ATable**[2], предназначенная для создания в чертеже произвольных таблиц посредством простого интерфейса, похожего на работу с Microsoft Word и Microsoft Excel. Программу такого класса самостоятельно написать сложно и ее могут приобрести многие. Кроме "маленьких" программ существует и ряд крупных систем, о которых мы упоминали во *Введении.*

Но у всех упомянутых больших и маленьких программ есть крупнейший недостаток — они являются приложениями к дорогой базовой системе AutoCAD. Любая САПР на базе AutoCAD не стоит ни гроша, если у пользователя нет самой системы AutoCAD. Система AutoCAD в России имеется почти у каждого проектировщика, за исключением работающих на иных базовых платформах. Сколько копий AutoCAD используется — никто не знает. Однажды мы встречали упоминание о 40 000 пользователей системы AutoCAD (было это лет десять назад, информация исходила из представительства фирмы Autodesk). Большинство этих пользователей, мягко говоря, являются незарегистрированными. Сколько продается легальных экземпляров AutoCAD выяснить, без "маски-шоу", даже у одного дистрибьютера, невозможно. Говорят, что много, а насколько много — два, десять экземпляров или десять тысяч, умалчивают. Предполагаем, что все-таки даже не сотни копий в год, иначе фирма Autodesk не потеряла бы интерес к России.

Эти рассуждения мы привели для того, чтобы стало понятно — пользователи пиратских версий базовой системы вряд ли будут покупать приложения к ней. Но в Рос-

---

[1] **www.cadsofttools.com**.

[2] **www.alx.ncn.ru**.

сии все не так, как во всем мире, и именно пользователи-пираты могут оказаться добросовестными покупателями *вашей* системы. Как это делается, рассмотрим чуть позже.

Составление бизнес-плана

Разработчики программ очень не любят составлять "ненужные" документы, к которым они часто относят и программную документацию, а тут еще какой-то бизнес-план. Зачем он, когда миллионы покупателей только и ждут появления нашей программы? К сожалению, все не так. Не будут миллионы пользователей вставать в очередь (как это бывало при выходе новых версий Windows). Попробуйте сначала объективно и без излишнего оптимизма оценить, сколько покупателей будет у вашей продукции — может оказаться, что и ни одного.

Заниматься бизнесом без плана можно в том случае, если вы распространяете простую и дешевую продукцию, примерами которой полны каталоги программ в Интернете. Если инсталляция программы "весит" до мегабайта, вы можете выложить ее на множество серверов для скачивания, а потом смотреть, что из этого выйдет. Если же инсталляция имеет размер несколько десятков мегабайт, то вряд ли вас куда-нибудь с таким "добром" пустят бесплатно, пользователи не захотят качать такого большого "кота в мешке" (Интернет у большинства не бесплатный), да и вам могут быть выставлены солидные счета от провайдера за чрезмерный исходящий трафик.

Мы не будем разбирать технологию составления бизнес-плана — этим вопросам посвящены многочисленные издания. Остановимся только на нескольких принципиальных вопросах, которые необходимо решить.

Цель вашего предприятия

От выбора конечной цели зависят и пути решения проблем. Решайте, чего вы хотите добиться — заработать миллион долларов, внедрить свою программу по всей стране, повысить производительность труда в своей организации или, может быть, "насолить" конкурентам. Конечно, хочется, чтобы вашей программой начали пользоваться все проектировщики бывшего СССР, но для начала поставьте более скромную задачу — "внедрить" ее в одной организации, но в полном объеме, во всех отделах, потом в масштабах города, а уж потом "слух о тебе пойдет по всей Руси великой". Мало желающих найдется делать ставку на "программу, которую сделал Вася Пупкин", а вот "система, успешно используемая в ЧегоТоТамПРОЕКТе" наверняка привлечет внимание. Если программа не используется в вашем родном городе, почему ее будут приобретать в других местах?

Кому нужна ваша программа

О том, кто может быть пользователем подобной программы, мы рассуждали в *главе 2.* Теперь необходимо определиться, кто может *купить* программу. Вопрос этот очень сложный. Сначала нужно определить количество потенциальных пользователей программы, не вдаваясь в их желания и возможности. По нашим подсчетам у программ такого класса может быть до 3000 организаций-пользователей на терри-

тории бывшего СССР, и это очень оптимистичный вариант. Реальных покупателей программы может быть около 10% от общего количества.

Замечание

Усложняет распространение и ситуация, с которой мы несколько раз сталкивались — в проектных институтах, для которых подобная программа очень нужна и высоко оценена пользователями, главными противниками ее приобретения становятся специалисты отделов САПР. Обычно они заявляют, что "такое мы и сами можем сделать". Иногда это правда (тогда говорят, что уже сделали), но чаще за этими заявлениями скрывается желание скрыть свою некомпетентность, которая неизбежно проявится при появлении чужаков на их территории.

Потенциальных пользователей САПР нужно разделить на несколько категорий:

- бывшие государственные проектные организации традиционного образца;
- проектные подразделения промышленных предприятий;
- проектные фирмы нового типа;
- частные лица.

С каждой из этих групп нужно вести собственную политику распространения программы. Большинство традиционных проектных организаций никогда не будут постепенно приобретать несколько копий или лицензий программы. Им нужно продавать сразу и для использования на любом количестве компьютеров. При этом не следует рассчитывать на оплату наличными — непременно потребуется прикрытие в виде солидного юридического лица, но и продать программу можно за большие деньги.

В стране появляется все больше частных проектировщиков, большинство из них работают дома на компьютерах, их благосостояние полностью зависит от результатов труда. Это очень перспективная категория пользователей, которых не надо особенно уговаривать — они сразу видят пользу для себя и готовы платить за "нематериальный" продукт, причем формы оплаты могут быть самыми разнообразными, иногда экзотичными (машина арбузов, бесплатный Интернет, аренда гаража). Именно такие пользователи обычно используют пиратские версии AutoCAD, но готовы оплачивать труд российских разработчиков.

Зачем нужна ваша программа

Для того чтобы вашу программу приобрели, потенциальные пользователи должны понять, зачем она им нужна. Общие фразы о повышении производительности и качества не всегда дают эффект. Если проектная организация не загружена заказами и не ликвидирована еще только потому, что это не допускается условиями приватизации, то зачем ей рост производительности? Но и в такой умирающей "конторе" могут быть отдельные группы, перегруженные "левыми" заказами. Это ваши потенциальные клиенты.

Организациям, не имеющим проблем с работой, надо объяснять, что ваша система позволяет автоматизировать работу над всеми разделами проекта, вводить единый стандарт проектирования, создавать упорядоченный электронный архив.

Попробуйте оценить, насколько повышается производительность. Лучше, если это будут не мифические "разы", а проценты повышения зарплаты пользователей.

Что вы будете продавать

Необходимо решить, а что именно физически вы будете продавать — коробки с дисками и документацией, лазерные диски, записанные на домашнем компьютере, или некие ключи, превращающие закачанную из Интернета программу в полнофункциональную. Рискнем предположить, что "коробочную" версию вы никогда не выпустите — сначала из-за отсутствия средств, а потом из-за того, что такой вид поставок отомрет вообще.

Продавать в принципе лучше не копии, а права на использование программы путем заключения *Договора о получении авторских прав на использование программы для ЭВМ конечным пользователем.*

Еще лучше продавать повышение производительности конкретных исполнителей, хотя сделать это намного сложнее.

Сколько стоит ваша программа

Разумеется, в рыночных условиях цена на любой товар определяется спросом и предложением, но необходимо знать, во что обошлось нам изготовление продукта.

В *главе 1* мы пытались определить трудозатраты на разработку программы, составившие, в первом приближении, 2 084 чел.-часов. Округлив эту цифру до 3 000, приняв среднюю часовую ставку программиста 80 рублей, накладные расходы 500% и "советскую" норму плановых накоплений 6%, мы получим примерные затраты на разработку программы 1 272 000 рублей.

> Вы думаете, что не затратили таких денег на разработку? Вы получали только небольшую зарплату? А кто же ее вам выплачивал, кто платил за аренду помещений, коммунальные услуги, Интернет, выплачивал многочисленные налоги? Если ничего этого не было, то программа разрабатывалась на необитаемом острове, а единственным ее пользователем станет Пятница. Но и Робинзон Крузо начинал не с пустыми руками, а "приватизировал" имущество с разбитого корабля.

В столице затраты на разработку будут еще больше, т. к. мы оперировали ставкой провинциального программиста. Теперь увеличим затраты еще в десять раз, учтя доведение программы до уровня товарного продукта, и полученную сумму в 12 720 000 рублей разделим на минимальное количество продаж — 300 инсталляций.

Стоимость инсталляции будет 42 400 рублей или примерно 1 400 долларов. Если вы думаете, что найдете в современной России за один год 300 покупателей продукта такого класса по такой цене, то очень ошибаетесь. Система такого класса должна для конечного пользователя иметь цену, не превышающую аналоги. Давайте рассмотрим хотя бы приблизительные аналоги. Для простоты воспользуемся сравнением с продуктами, распространяемыми уважаемой фирмы Consistent Software (табл. 37.1).

Напомним, что мы рассуждаем о *вашей* системе, имеющей возможности не ниже, чем у ruCAD.

Мы выбрали три очень хороших и популярных продукта. Project Studio CS — это "потомок" давно известной системы АРКО, ранее разрабатывавшейся и распространявшейся московской проектной фирмой АПИО-Центр, МАЭСТРО — продукт Maestro Group, распространяемый Consistent Software в России, СПДС GraphiCS —

удобный инструментарий, закрывающий бреши в продуктах фирмы Autodesk по оформлению чертежей в соответствии со стандартами СПДС. К сожалению, последний продукт появился лет на пятнадцать позже, чем в нем возникла потребность.

Таблица 37.1. *Цены аналогичных программных продуктов*

Продукт	Возможности	Цена
СПДС GraphiCS 2.5	Набор инструментов архитектурно-строительной графики для оформления чертежей в соответствии с требованиями ГОСТ в среде системы AutoCAD 2002/2004, ADT 3.3/2004. Сертифицирован Госстроем России	$450
Project Studio CS v.1 Архитектура	Создание объемной модели и формирования полного комплекта рабочей документации архитектурного раздела проектирования (АС, АР, АИ) в среде AutoCAD 2002/2004/ADT	$900
МАЭСТРО-АКС (архитектура, конструкции, сантехника)	Приложение к системе AutoCAD 2002/2004. Сертифицирована Госстроем России	$1530

Соревноваться в бизнесе с этими продуктами тяжело. Они и сами "раскручены" и распространяются ведущей фирмой в области САПР. Следовательно, про все наши ценовые расчеты нужно забыть. При попытке влезть на рынок вы непременно столкнетесь с этими продуктами, и должны будете еще доказать потенциальным покупателям, что нужно не просто купить ваш продукт, но еще и купить его, например, вместо СПДС GraphiCS, и возможно, в дополнение к Project Studio CS.

Для этого есть немало оснований. Напомним, что вы предоставляете средства для автоматизации вычерчивания всех разделов проекта, и это очень важно. Вы много раз наблюдали, как мучаются с "голой" системой AutoCAD все смежники, хотя архитекторы и строители много лет работают в АРКО. Вы предусмотрели много программных решений, позволяющих расширять нашу систему, вы имеете удобный интерфейс с иллюстрированными меню, классификатор слоев, достаточно "продвинутые" элементы документооборота и многое другое. Ваша система полностью совместима[1] с рассматриваемыми продуктами и может даже поглотить их путем включения в собственные иллюстрированные меню. Пользователи смогут выбирать между оформительскими средствами СПДС GraphiCS и вашими, и еще неизвестно, какие окажутся популярнее.

Прежде чем решить, как выжить в соревновании с такими конкурентами, мы дополнительно "пострашаем" читателей некоторыми юридическими проблемами, требующими решения.

Юридические вопросы

Просмотрите еще раз *главу 1*, в которой мы советовали как можно раньше решить вопросы авторского права. Теперь, при завершении работы, вы уже должны не

[1] В отличие от систем, рисующих модель в масштабе, а не в натуральную величину.

только точно знать ответы на все вопросы авторского права, но иметь и документы, в которых все это подтверждено — от соглашения между соавторами до соглашений с организацией, в которой вы работаете. Необходимо хорошо изучить упоминавшиеся нами законы и действовать в точном соответствии с ними, не забывая, что в России, кроме законов, действуют еще и "понятия".

Авторские права

Четко установите владельцев *личных* и *имущественных* прав — личные авторские права у вас никто не отнимет (если соавторы не перессорятся, "деля шкуру неубитого медведя"), а имущественные права можно продать. Вообще-то, если вы не чувствуете в себе талантов коммерсанта, самым лучшим вариантом для разработчиков является именно продажа имущественных прав, вопрос только кому и за сколько. Не стоит проявлять в этом вопросе излишних амбиций — для доведения ваших программ до уровня продаваемого продукта потребуется средств намного больше, чем на разработку. Правда и покупателей на такое "имущество" сейчас найти очень трудно, т. к. фирмы-идеалисты, покупавшие интеллектуальную собственность лет 15 назад, сгинули в это смутное время, а выжившие еще не прошли путь от бизнеса на "железе" до продвижения российских продуктов "вне фирменных" разработчиков. Взять "на реализацию" готовый продукт — это можно, а доводить до товарного вида посторонний — увольте.

Регистрация программ

После изучения законов у вас отпадут и вопросы о "патентовании" программ. Авторы имеют право на регистрацию программы для ЭВМ и базы данных, но не обязаны это делать. Надо знать и то, что законом охраняются не идеи и принципы организации интерфейса и алгоритмы, а лишь конкретная реализация этого алгоритма в виде совокупности данных и команд. Регистрация программ и баз данных осуществляется Роспатентом в соответствии с *Правилами составления, подачи и рассмотрения заявки на официальную регистрацию программы для электронных вычислительных машин и заявки на официальную регистрацию базы данных*[1], утвержденными приказом Роспатента от 25 февраля 2003 года № 25.

Сертификация и лицензирование

Иногда возникают вопросы о сертификации и лицензировании программ. Действительно, имеются предметные области, в которых сертификация программ является обязательной, а фирмы-разработчики должны иметь лицензию на соответствующую деятельность. Программы класса "чертилка", рассматриваемые в книге, сертификации не подлежат. Другое дело, что солидная (вернее, богатая) фирма может правдами и неправдами получить, "для понту", сертификат на программный продукт, но наличие такого сертификата свидетельствует только о том, что "поддержка отечественному чиновнику" оказана. Самую лучшую "сертификацию" осуществит пользователь.

---

[1] **www.fips.ru/avp/pr_sw_db.htm**.

Юридическое лицо

Далее следует решить вопрос с "юридическим лицом". Конечно, автор имеет полное право (если он имеет имущественное право) торговать своей продукцией самостоятельно — хоть на рынке. При этом сразу встанет вопрос о легализации этой деятельности, как минимум, в качестве частного предпринимателя без образования юридического лица. Но особенность САПР в том, что основными покупателями являются организации, а они привыкли иметь дело с другими организациями. Вот тут-то и может пригодиться ваша работа в проектной организации — оформляйте внутренние взаимоотношения, делитесь доходами и работайте "под крышей". Возможен вариант ухода "под крышу" какой-нибудь солидной компьютерной фирмы — на поверку многие из них, имея солидную марку, оказываются неким товариществом предпринимателей или лиц без определенных (по трудовой книжке) занятий.

Проверка собственных прав

Если вы намечаете коммерческую деятельность, то реализуемая вами продукция должна быть изготовлена с использованием лицензионных программных средств. Прежде всего это, конечно, наша базовая система — AutoCAD. Лицензионными должны быть и операционная система, и используемые системы программирования (Delphi, VC++), и все компоненты и библиотеки, примененные при разработке. Не забудьте включить затраты на все это "добро" (не менее $10 000) в свой бизнес-план.

Лицензионные соглашения

Для завершения юридических вопросов подготовьте лицензионное соглашение на вашу программу. При продаже и предоставлении массовым пользователям доступа к программам законодательством допускается применение особого порядка заключения договоров, например, путем изложения типовых условий договора на передаваемых экземплярах программ —"оберточных" лицензиях (shrinkwrap license). В отличие от традиционных авторских договоров, заключаемых в предписанной законодательством письменной форме, данные договоры являются разновидностью сделок, при которых лицо выражает свою волю установить правоотношение не в форме устного или письменного волеизъявления, а своим поведением, по которому можно сделать заключение о таком намерении — вскрытие упаковки или выполнение программы установки после знакомства с договором.

Составлять лицензию нужно тщательно, с привлечением юристов. Мы можем только посоветовать:

- называть лицензию *Договором о получении конечным пользователем авторских прав на использование программы для ЭВМ*, при таком названии возможны льготы по налогам, касающихся "получения авторских прав";
- не применять для своих программ лицензионные соглашения иностранных поставщиков — они могут не соответствовать российскому законодательству.

Подведем итоги

Если читатели напуганы сложностями с распространением собственных программ (и юридические "крючки", и конкуренция, и большие затраты на подготовку), то не переживайте — многого мы не упомянули, ограничившись намеком на "среднепотолочное" десятикратное увеличение стоимости подготовки товарного продукта, по сравнению со стоимостью разработки программы. Решать нужно еще множество вопросов — от дизайна обложки компакт-диска до организации собственного сайта для технической поддержки продукта и технологии распространения программ класса Shareware. Этим вопросам посвящено множество специализированных изданий и ресурсов в Интернете.

Понимая, что упомянутые проблемы сразу решить трудно, мы советуем не распылять силы и последовательно действовать в соответствии с планом:

- внедряйте программу версии 1.*xx* сначала в собственной фирме, а потом и в других проектных организациях города;
- начинайте переговоры о внедрении не с руководителями организаций и не с работниками САПР, а с заинтересованными специалистами производственных отделов;
- используйте собственную программу для разработки проектов — если вы уверяете, что с ее помощью "чертить в десять раз быстрее", так докажите это на деле, увеличив хотя бы в два раза свою зарплату;
- передавайте программу бесплатно, но предлагайте платную техническую поддержку и обучение — в масштабах города это сделать проще;
- предлагайте оплачивать свой труд в процентах от повышения зарплаты исполнителей, использующих программу;
- выявляйте ошибки и неудобные пользователям интерфейсы, собирайте статистику по использованию команд для размещения самых популярных на панелях инструментов;
- обучите нескольких активистов приемам пополнения библиотек блоков, типовых текстов, типовых проектных решений и пополняйте программу этими данными;
- организуйте общий банк данных оборудования и изделий для спецификаций;
- передавайте программу в вузы и техникумы для обучения и выполнения курсовых и дипломных работ — студенты быстро растащат ее по домам и по всему свету;
- постепенно создавайте свой сайт в Интернете, наполняя его до открытия реальными вопросами и ответами, иллюстрирующими техническую поддержку программы;
- размещайте на сайте достаточно подробную документацию;
- начинайте постепенно рекламировать программу в Интернете, но никогда не рассылайте рекламные письма, не употребляйте выражения "лучшая программа", "единственный продукт" и т. п.;
- выделите отдельные интересные программы для работы в "автономном" режиме и разместите их в различных коллекциях (упоминая, что это маленький фрагмент

продукта) — бесплатно с русскоязычным интерфейсом и в качестве условно-бесплатных[1] с англоязычным интерфейсом.

Примерно через год работы в таком режиме, после устранения явных огрехов, усовершенствования программы, начинайте внедрение версии 2.*xx* и в других регионах.

Еще через год, набравшись опыта и подготовив существенно усовершенствованную версию 3.*xx*, составляйте бизнес-план и приступайте к реализации коммерческих замыслов. Если они еще останутся.

[1] Прежде, чем это делать, разузнайте, как это делается, например на сайте **www.softshape.com/swrus**. Здесь вы найдете все материалы и ссылки по "шароварной" тематике.

ГЛАВА 38

Особенности AutoCAD 2005

В задании на разработку нашей системы был предусмотрен переход на систему AutoCAD 2004, а в *главе 4* приведены особенности AutoCAD 2004. Пока мы разрабатывали нашу систему, фирма Autodesk выпустила новую версию — AutoCAD 2005. Некоторые его особенности мы упоминали в предыдущих главах, теперь рассмотрим эту версию подробнее.

В целом система AutoCAD 2005 сразу производит очень хорошее впечатление. В ней появились и радикальные улучшения, и множество приятных мелочей.

Мы не будем детально описывать новые возможности, которые получили пользователи — для этого требуется отдельная книга, а наша посвящена программированию. Упомянем только основные новинки:

- Sheet Set Manager — новый инструмент управления проектами;
- команда TABLE — удобное средство для создания и заполнения таблиц;
- команда FIELD — превосходное средство, позволяющее вставлять в рисунок автоматически обновляющуюся текстовую информацию (около сорока типов, в том числе и пользовательские свойства рисунка, и значения системных переменных);
- усовершенствованы TOOL PALETTES (ИНСТРУМЕНТАЛЬНЫЕ ПАЛИТРЫ) — появилась возможность добавлять собственные макросы, как и в панели инструментов;
- улучшена работа со слоями — появилась возможность группировки слоев и управления этими группами (к сожалению, дерево групп слоев ограничено одним уровнем).

Изменено множество команд и системных переменных, появились 25 новых команд и много новых системных переменных.

Новинки для разработчиков

Нас в AutoCAD 2005 интересует прежде всего то, как сможет работать с ним наша система. В каждой новой версии мы сталкивались с какими-то неприятными сюрпризами, о которых, в отличие от приятных, фирма Autodesk скромно умалчивает. Разумеется, для обнаружения всех "подводных камней" требуется долгое и тщатель-

ное тестирование в различных режимах, но некоторые предварительные выводы можно сделать сразу.

Хорошие новости

Главная и самая приятная новость — после установки системы AutoCAD 2005 наша программа-стартер обнаружила ее, запустила, и заработали (насколько это можно оценить сразу) все наши программы.

Проверка некоторых критичных мест показала, что в новой версии сохранились и некоторые "глюки" системы AutoCAD 2004. Так как мы их уже преодолели, то это можно считать лучшим вариантом, чем их "усовершенствование".

Формат откомпилированных файлов меню остался совместимым с версией 2004, и это позволяет не создавать для AutoCAD 2005 отдельную папку на путях поиска файлов поддержки.

Плохие новости

Иногда возникают проблемы с установкой системы AutoCAD 2005. Прежде всего, на некоторых компьютерах может возникнуть ошибка установки среды .NET Framework (рис. 38.1).

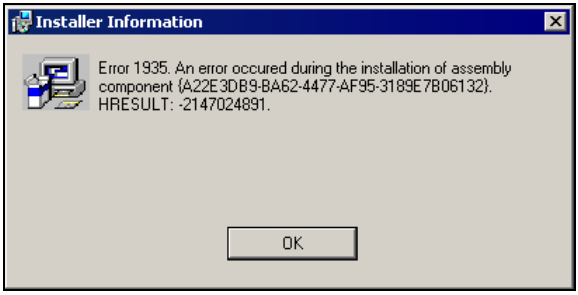


Рис. 38.1. Сбойная ситуация

Эта ошибка к системе AutoCAD отношения не имеет и может возникнуть на компьютерах с ранее установленной средой .NET Framework. Излечивается эта ошибка отключением на время инсталляции всех антивирусных мониторов. Возможно, потребуется удаление Internet Information Service[1] (IIS), если он зачем-то был установлен.

Следующая ошибка может возникнуть при запуске acad.exe (рис. 38.2).

Сообщение говорит о том, что модуль acad.exe обратился не к той библиотеке acdb16.dll. Скорее всего была найдена первой версия библиотеки от AutoCAD 2004 в c:\Program Files\Common Files\Autodesk Shared\acdb16.dll. В этой DLL действительно нет функции `evaluationStatus`, она должна быть в одноименной DLL для 2005. Это

[1] Это всего лишь слухи, но некоторые клянутся, что помогало.

означает, что система AutoCAD 2005 была установлена не полностью. Многие компоненты версии 2005 устанавливаются в ту же паку, что и для версии 2004. Чаще всего это происходит правильно, но возможны и сбои. Для лечения можно попробовать на время установки AutoCAD 2005 переименовать папку c:\Program Files \Common Files\Autodesk Shared.

Рис. 38.2. Ошибка при запуске acad.exe

Если система AutoCAD 2005 установилась и запускается (обычно так и происходит), то надо проверить работу предыдущих версий. Это только в фирме Autodesk полагают, что перед установкой новой версии будут удаляться старые, мы-то с читателями знаем, что это не так. К счастью, система AutoCAD 2004 работает с библиотеками версии 2005, как со своими[1], и это хорошо.

Оставлять в работе систему AutoCAD 2004 при установленной AutoCAD 2005 имеет смысл только для искателей приключений и разработчиков, просто обязанных тестировать программы в разных версиях базовой системы. Именно этим мы занимаемся, у нас предусмотрена возможность запуска разных версий системы AutoCAD (по алгоритму, описанному в *главе 20*), и мы это должны проверить. Вот тут и начинаются более серьезные неприятности.

Каждый acad.exe, как приложение, запускается и работает нормально, но нас интересуют более глубокие вопросы. Напомним, что делает наша программа-стартер:

- разыскивает все установленные версии AutoCAD;
- для каждой версии определяет полный путь к файлу acad.exe;
- для каждого acad.exe выясняет имя объекта автоматизации `AcadApplication` (`AutoCAD.Application`, `AutoCAD.Application.15` или `AutoCAD.Application.16`);
- первоначальный запуск любой из выбранных систем AutoCAD выполняет как для обычного приложения, загружая требуемый acad.exe;
- при запущенной системе AutoCAD обращается к ней как к объекту класса `OleVariant`, посылая ей соответствующие указания — обычно открыть файл.

Для связи с объектом автоматизации используется вычисленное имя.

После того, как в реестре Windows похозяйничала система AutoCAD 2005, обнаруживаются удивительные вещи.

Напомним, что в реестре имеются несколько ветвей:

```
HKEY_CLASSES_ROOT\AutoCAD.Application.14\CLSID
HKEY_CLASSES_ROOT\AutoCAD.Application.15\CLSID
HKEY_CLASSES_ROOT\AutoCAD.Application.16\CLSID
```

[1] Предполагаем, что до поры до времени.

В каждой из этих ветвей имеется ключ "По умолчанию", а значением ключа является идентификатор этой версии системы AutoCAD, по которому можно найти остальные параметры, в том числе полное имя EXE-файла, и сопоставить каждому файлу acad.exe соответствующее имя объекта автоматизации.

Особую роль играет ветвь `HKEY_CLASSES_ROOT\AutoCAD.Application`, в которой, кроме подветви `CLSID`, имеется подветвь `CurVer`. Сюда записываются параметры последней запущенной системы AutoCAD (изменение происходит после полной загрузки acad.exe). Например, после запуска AutoCAD R14 фрагмент реестра будет выглядеть, как показано в листинге 38.1, после запуска AutoCAD 2002 — как в листинге 38.2, а после запуска AutoCAD 2004 — как в листинге 38.3.

Листинг 38.1. Ветвь реестра `AutoCAD.Application` после запуска AutoCAD R14

```
[HKEY_CLASSES_ROOT\AutoCAD.Application]
@="AutoCAD Application"
[HKEY_CLASSES_ROOT\AutoCAD.Application\CLSID]
@="{D2F13240-5769-11CF-A820-0800091B9B14}"
[HKEY_CLASSES_ROOT\AutoCAD.Application\CurVer]
@="AutoCAD.Application.14"
```

Листинг 38.2. Ветвь реестра `AutoCAD.Application` после запуска AutoCAD 2002

```
[HKEY_CLASSES_ROOT\AutoCAD.Application]
@="AutoCAD Application"
[HKEY_CLASSES_ROOT\AutoCAD.Application\CLSID]
@="{8E75D911-3D21-11d2-85C4-080009A0C626}"
[HKEY_CLASSES_ROOT\AutoCAD.Application\CurVer]
@="AutoCAD.Application.15"
```

Листинг 38.3. Ветвь реестра `AutoCAD.Application` после запуска AutoCAD 2004

```
[HKEY_CLASSES_ROOT\AutoCAD.Application]
@="AutoCAD Application"
[HKEY_CLASSES_ROOT\AutoCAD.Application\CLSID]
@="{1365A45F-0C8F-4806-A26A-6B22AD37EC66}"
[HKEY_CLASSES_ROOT\AutoCAD.Application\CurVer]
@="AutoCAD.Application.16"
```

Такая система позволяет сторонним приложениям в любой момент определить истинное название объекта автоматизации `AutoCAD.Application`.

После запуска AutoCAD 2005 эта система оказывается поломанной. AutoCAD 2005 числится в реестре как `AutoCAD.Application.16.1` и, разумеется, имеет свой CLSID:

```
[HKEY_CLASSES_ROOT\AutoCAD.Application.16.1\CLSID]
@="{FC280999-88C6-4499-9622-3B795A8B4A5F}"
```

Однако после запуска, вместо того, чтобы сделать запись (листинг 38.4), как это делали ее "предки", эта наглая система влезает в чужой ключ, идентификатор которого принадлежит AutoCAD 2004, и записывает туда собственное имя файла (листинг 38.5).

Листинг 38.4. Как должна была регистрироваться система AutoCAD 2005

```
[HKEY_CLASSES_ROOT\AutoCAD.Application\CLSID]
@="{ FC280999-88C6-4499-9622-3B795A8B4A5F }"
[HKEY_CLASSES_ROOT\AutoCAD.Application\CurVer]
@="AutoCAD.Application.16.1"
```

Листинг 38.5. Запись AutoCAD 2005 в ключ AutoCAD 2004

```
[HKEY_CLASSES_ROOT\CLSID\{1365A45F-0C8F-4806-A26A-6B22AD37EC66}\LocalServer32]
@="C:\\Acad\\2005\\acad.exe /Automation"
```

Это означает, что при обращении к идентификатору AutoCAD 2004 будет происходить подмена на файл иной версии.

Что делать?

А какое нам дело до внутренних махинаций фирмы Autodesk с ее ключами реестра? Может быть, это у них новая политика, в которой мы ничего не понимаем, а может быть, просто кто-то что-то не оттуда, откуда надо, скопировал и не туда, куда надо, вставил? Бывает ведь и "кое-где, у нас порой". Конечно, бывает со всеми, тем более что если отследить приключения идентификаторов во всем реестре, то становится понятно, что никакой "новой политики" нет, просто ошибки.

Для нашей системы это имеет прямые последствия. Наша программа-стартер ищет объект автоматизации так, как положено — по идентификатору, а натыкается на EXE-файл, не относящийся к этому идентификатору. В результате у нас происходит конфликт имен, и мы не сможем "по науке" работать с требуемым объектом автоматизации. Это означает, что мы не сможем отправить в работающую систему AutoCAD новый файл, будем запускать новые процессы и вообще будем вынуждены работать не так, как положено. Конечно, можно и отыскать работающую систему AutoCAD в списке процессов, но это будет возврат на несколько лет назад.

Поразмыслив, мы приняли решение откорректировать программу-стартер в сторону ухудшения ее возможностей. Изменять концепцию нет смысла — вероятность того, что в фирме Autodesk опомнятся и исправят ошибку, есть. Пока мы просто заблокируем возможность загрузки дополнительных файлов в систему AutoCAD из стартера (это не так уж и нужно) и, для памяти, будем выводить в рабочем окне напоминание о конфликте версий. Вдаваться в детали, как это сделано, мы уже не будем.

Резюме

Новая версия системы AutoCAD 2005 является интересной и прогрессивной. Некоторое количество ошибок[1] простительно в любом продукте, ошибки преодолимые, и на AutoCAD 2005 мы и будем ориентироваться в дальнейших версиях ruCAD.

[1] Еще более "обнадеживающий" список "багов" можно найти на **www.jtbworld.com**.

После дополнительных исследований новых объектов мы включим в свою систему дополнительные возможности. Просто напрашивается автоматизированное формирование Tool Palettes из наших XML-меню, тем более что наши иллюстрации к меню "проглатываются" этими палитрами с удовольствием. В отличие от стандартных панелей инструментов, для которых самым трудоемким является ручное рисование миниатюр, Tool Palettes работают с файлами, а это позволит, кроме XML-меню, сформировать и полный набор палитр. Но это все в будущем.

Послесловие

Итак, мы завершаем эту трудную и для нас, и для читателей книгу. Мы впервые описывали, а читатели, возможно, впервые читали, как делаются программы — не справочные сведения, не отдельные примеры, не изучение языка, а технологический процесс создания программного продукта.

Трудности для авторов были в том, что мы делали реальную систему, которую можно использовать в реальной работе, да еще и описывали, как это делается. Трудности для читателей мы видим в том, что описывались не всем знакомые технологии, да еще с минимальным разъяснением азов. Но система реальная, и если мы использовали для разработки ее компонентов Borland Delphi, то мы и посвятили значительную часть книги изучению связки Delphi—AutoCAD. Не нашлось в нашей системе применения для VBA — мы и не стали внедрять в книгу искусственных примеров. Впрочем, примеров использования VBA полна справочная система AutoCAD.

Подведем некоторые итоги выполнения задач, поставленных в начале книги.

Сроки разработки

В *главе 1* мы подсчитали примерную трудоемкость работ по разработке такой системы — 260,5 чел.-дней, и заявили, что это вполне реально. Конечно, такая точная расчетная трудоемкость (с трогательным "полднем") рассчитана на "облапошивание" заказчика, т. к. вызывает большее доверие, чем, например, "примерно год", хотя, по сути, это и есть год. У нас, кстати, был реальный заказчик, который хорошо разбирается в том, как составляются планы и как они реально выполняются.

Теперь мы знаем, что хотя и, разумеется, не выдержали эти "полдня", но в "примерно год" уложились. Заказчик этим был очень удивлен, т. к. думал, что фактически мы будем возиться не менее трех лет. Вообще-то мы уложились бы и в запланированные сроки, но одновременно с разработкой программ мы писали и книгу. Если приравнять ее к программной документации, то со сроками у нас получается почти идеальная картина[1]. А если бы мы делали систему для одной версии AutoCAD, то могли бы сделать еще быстрее — не менее 10% времени было потрачено на разборки "с сюрпризами" фирмы Autodesk в различных версиях базовой системы.

---

[1] В отличие от сроков выпуска книги, но мы пошли на их увеличение, чтобы в книге был отражен реальный процесс разработки и реальные результаты.

Результаты разработки

В *главе 2* мы изложили основные технические требования к системе, а в *главе 3* — базовые принципы, включая стандарт работы в нашей САПР. Все эти планы выполнены, хотя пришлось вносить и некоторые коррективы.

С коммерческой точки зрения мы совершили типичную ошибку российских разработчиков — сделали систему лучше, чем требовалось, вместо того, чтобы быстрее "пустить в оборот" то, что есть. Например, мы могли бы спокойно использовать давно отработанные меню в формате BTR, не заниматься исследованиями XML, оставить все старые программы с использованием функции `command`, без переделки на объектные технологии и оптимизации исходных текстов. Пользователи системы ничего бы об этих технических деталях не знали, да и книга давно бы уже вышла из печати. Но мы посчитали невозможным приводить в книге устаревшие решения и работающие, но некрасивые программы. В этом мы солидарны со многими проектировщиками, не выпускающими "сырых" проектов, как бы на них не нажимало начальство — о том, что "быстро делали", забудется уже завтра, а репутация специалиста может быть испорчена навсегда.

Как и большинство разработчиков, мы не выполнили клятву — все сразу документировать, хотя и пытались это делать. Мы даже создали проекты справочной системы и по функциям ruCAD, и по командам. Увы, не располагая специальным "дивизионом" для таких работ, мы, как всегда, отложили эти работы "на потом", хотя и предусмотрели средства, облегчающие подготовку документации — текстовые и HTML-справки и тому подобные суррогаты.

Параметры книги

К сожалению, не все, что было задумано, удалось вместить в ограниченный объем издания — прежде всего это касается примеров прикладных программ. Намного больше, чем планировалось, заняли различные, иногда спорные, рассуждения. Мы пошли на это по советам наших консультантов — побольше разъяснять, почему принято то или иное решение, а исходные тексты можно всегда просмотреть на компакт-диске.

Мы не можем считать идеальными все приведенные функции — наверняка читатели смогут найти более красивые решения. Можно считать недостатком смесь стилей — встречаются и функция `command`, и функция `entmake`, и работа с использованием механизмов ActiveX. Может вызвать удивление публикация множества примитивных и неинтересных функций. Все это объясняется одним — мы описывали создание реальной системы в реальных условиях, а реальное изделие, как правило, не состоит из одних гениальных конструкций — в нем встречаются и обычные болты, и гайки.

Мы писали книгу о том, *как делается* САПР, а не о том, как она *должна делаться*. Мы живем так, как живем, а не так, как должны жить, и программируем так, как программируем, а не так, как должны программировать. Для того чтобы понять, как надо "жить", нужно "задокументировать" существующий "образ жизни". Это мы и попытались сделать.

Получилось это или нет — судить нашим читателям.

ПРИЛОЖЕНИЕ

Описание компакт-диска

В книге мы неоднократно упоминали о дополнительных материалах, которые будут размещены на прилагаемом компакт-диске. Однако сформировать диск оказалось не так просто, и не потому, что у нас недостаточно материалов, а потому, что их слишком много и описание прилагаемых материалов может занять несколько десятков страниц. В результате мы решили схитрить и на компакт-диск поместить минимум "единиц хранения", но предоставив читателям возможность ознакомиться с максимумом материалов.

Итак, на компакт-диске в папке **Install** находятся две программы для установки системы:

- ruCADBookLT.exe — установка "книжной" версии системы ruCAD;
- ruCADSource.exe — установка исходных текстов системы и листингов.

Такую технологию мы применили для того, чтобы читатели, прочитав в книге, *как это делается*, могли увидеть и результат работы, то есть *как это сделано.*

Как устанавливать

Для установки материалов на компьютер необходимо запустить соответствующую программу установки.

При запуске программ установки будет выведено для прочтения и подтверждения лицензионное соглашение. Соглашаясь с лицензионным соглашением, вы не просто нажимаете кнопку, а "подписываете" договор, про который не сможете сказать, что "в глаза его не видели"[1]. В частности, в лицензионном соглашении указано, что **прилагаемые исходные тексты вы имеете право использовать и для создания коммерческих программных продуктов, но только при условии поставки их с исходными текстами**, а для разработки "закрытых" программных продуктов требуется приобретение соответствующей лицензии.

---

[1] Еще одна причина создания установочной программы, а не записи материалов на компакт-диск "россыпью".

Что будет установлено

При запуске программы **ruCADBookLT.exe**[1] будут выполнены действия, предусмотренные сценарием установки, описанным в *главе 36*. В результате работы инсталлятора будет установлена специальная "книжная" версия системы ruCAD. Мы предоставляем читателям возможность самостоятельно разобраться (по книге и прилагаемому исходному тексту сценария установки), что и куда будет поставлено.

"Книжная" версия является полностью работоспособной, в ней нет никаких ограничений, например на количество запусков. От рабочей версии она отличается только тем, что в нее входят только отдельные прикладные программы. "Книжная" версия системы предназначена для иллюстрирования решений, описанных в книге, а не для изготовления проектной продукции.

При необходимости "книжную" версию можно обновить до рабочей, скачав с сайта **www.kurganobl.ru/cad** необходимые дополнения. Дополнения выкладываются в виде небольших пакетов обновления (Service Pack), в которые, как правило, включаются FAS-файлы прикладных программ и, при необходимости, дополнительные компоненты (меню, библиотеки блоков).

При запуске программы **ruCADSource.exe** на компьютер пользователя будут установлены все исходные тексты, включая листинги, приводившиеся в книге (в полном объеме и с восстановленным форматированием), LSP-, PAS-, DFM- и DPR-файлы и прочие исходные материалы, "разложенные по полочкам" в систему каталогов.

Кроме того, будет установлена специальная программа-навигатор для удобного просмотра исходных текстов с иллюстрациями и описаниями файлов. Используя программу-навигатор, вы сможете перетаскивать выбранные файлы в среду разработки Visual LISP или Delphi для более детального знакомства.

Что не входит в установочный комплект

Не входят в комплект установки исходные тексты сторонних библиотек функций и компонентов. Хотя мы и используем только бесплатные библиотеки и компоненты, поставляемые с исходными текстами, но распространять их не имеем права. Ссылки на сайты с компонентами мы приводили в книге, при затруднениях можно обратиться на сайт **www.kurganobl.ru/cad**, на котором мы отслеживаем "места прописки" используемых библиотек.

[1] Инсталляция возможна только при наличии AutoCAD R15 и старше.

Источники информации

Список литературы

1. David M. Stein. The Visual LISP Developers Bible — 2003 Edition. Электронное издание, 2003. **www.dsxcad.com**. — 183 с.
2. Архангельский А. Я. Программирование в Delphi 6. — М.: ЗАО "Издательство БИНОМ", 2001. — 1120 с.
3. Байбара В. А., Заболоцкий Д. В., Кнеллер М. И., Усвятцев О. Б. AutoCAD. Полезные рецепты. — М.: Радио и связь, 1994. — 208 с.
4. Бугрименко Г. А., Лямке В. Н., Шейбокене Э.-К. С. Автоматизация конструирования на ПЭВМ с использованием системы AutoCAD. — М.: Машиностроение, 1993. — 336 с.
5. Гинцбург Э. Я. Расчет отопительно-вентиляционных систем с помощью ЭВМ. — М.: Стройиздат, 1979. — 183 с.
6. Гладков С. А. Программирование на языке Автолисп в системе САПР Автокад. — М.: ДИАЛОГ-МИФИ, 1991. — 96 с.
7. Горохов В. Ф., Мущанов В. Ф., Югов А. М. и др. Алгоритмы расчета стальных конструкций. — М.: Стройиздат, 1989. — 368 с.
8. Гофман В. Э., Хомоненко А. Д. Работа с базами данных в Delphi. — СПб.: БХВ-Петербург, 2001. — 656 с.
9. Дейт К. Дж. Введение в системы баз данных, 7-е издание: Пер. с англ. — М.: Издательский дом "Вильямс", 2001. — 1072 с.
10. Джамп Д. AutoCAD. Программирование: Пер. с англ. — М.: Радио и связь, 1992. — 336 с.
11. Жарков С. В. Shareware: профессиональная разработка и продвижение программ. — СПб.: БХВ-Петербург, 2003. — 320 с.
12. Зингер Н. М. Гидравлические и тепловые режимы теплофикационных систем. — 2-е изд., перераб. — М.: Энергоатомиздат, 1986. — 320 с.
13. Идельчик И. Е. Справочник по гидравлическим сопротивлениям. — М.: Госэнергоиздат, 1960. — 465 с.
14. Ильин Ю. А. Расчет надежности подачи воды. — М.: Стройиздат, 1987. — 320 с.

15. Ионин А. А. Надежность систем тепловых сетей. — М.: Стройиздат, 1989. — 268 с.

16. Ковязин А., Востриков С. Мир Interbase. Архитектура, администрирование и разработка приложений баз данных в Interbase/Firebird/Yaffil. — М.: КУДИЦ-ОБРАЗ, 2002. — 432 с.

17. Кречко Ю. А. AutoCAD: программирование и адаптация. — М.: ДИАЛОГ-МИФИ, 1995. — 240 с.

18. Круглински Д. Основы Visual C++: Пер. с англ. — М.: Издательский отдел "Русская редакция", 1996. — 696 с.

19. Курганов А. М., Федоров Н. Ф. Гидравлические расчеты систем водоснабжения и водоотведения. — Л.: Стройиздат, 1986. — 440 с.

20. Одельский Э. Х., Каган Л. И., Кирзнер Л. Х. Расчет систем центрального отопления и вентиляции на электронных вычислительных машинах. — Минск.: Вышэйшая школа, 1974. — 240 с.

21. Питц-Моултис Н., Кирк Ч. XML: Пер. с англ. — СПб.: БХВ-Петербург, 2000. — 736 с.

22. Платт Д. С. Знакомство с Microsoft .NET: Пер. с англ. — М.: Издательско-торговый дом "Русская редакция", 2001.

23. Полещук Н. Н. AutoCAD 2002. — СПб.: БХВ-Петербург, 2003. — 1200 с.

24. Полещук Н. Н. AutoCAD 2004. — СПб.: БХВ-Петербург, 2004. — 976 с.

25. Полещук Н. Н. AutoCAD 2004: разработка приложений и адаптация. — СПб.: БХВ-Петербург, 2004. — 624 с.

26. Полещук Н. Н. Visual LISP и секреты адаптации AutoCAD. — СПб.: БХВ-Петербург, 2001. — 576 с.

27. Роджерсон Д. Основы COM: Пер. с англ. — М.: Издательско-торговый дом "Русская редакция", 2000. — 400 с.

28. Свет В. Л. AutoCAD: язык макрокоманд и создание кнопок. — СПб.: БХВ-Петербург, 2004. — 320 с.

29. Степанова Т. А. Автоматизированное проектирование систем отопления. — Л.: Стройиздат, 1986. — 170 с.

30. Тейксейра С., Пачеко К. Delphi 5. Руководство разработчика, том 2. Разработка компонентов и программирование баз данных: Пер. с англ. — М.: Издательский дом "Вильямс", 2001. — 992 с.

31. Финкельштейн Э. AutoCAD 2000. Библия пользователя: Пер. с англ. — М.: Издательский дом "Вильямс", 2000. — 1040 с.

32. Фоменко В. Я., Любимцев А. Н., Любимцева С. Н. Русско-английский разговорник для строителей. — М.: Рус. яз., 1990. — 507 с.

33. Хармон Э. Разработка COM-приложений в среде Delphi: Пер. с англ. — М.: Издательский дом "Вильямс", 2000. — 464 с.

34. Хювёнен Э., Сеппянен Й. Мир Лиспа. В 2-х т. Т. 1: Введение в язык Лисп и функциональное программирование. Пер. с финск. — М.: Мир, 1990. — 447 с.

35. Хювёнен Э., Сеппянен Й. Мир Лиспа. В 2-х т. Т. 2: Методы и системы программирования. Пер. с финск. — М.: Мир, 1990. — 319 с.

36. Черносвитов А. Visual C++ и MFC. Курс MCSD для профессионалов. — СПб.: ПИТЕР, 2000. — 544 с.
37. Янг М. XML. Шаг за шагом: Пер. с англ. — М.: ЭКОМ, 2000. — 384 с.

Ресурсы в Интернете

В сети Интернет можно найти тысячи сайтов, посвященных работе в системе AutoCAD. Мы приведем только краткий список наиболее интересных (на наш субъективный взгляд), на которых освещаются вопросы, близкие к теме книги. В список не включены корпоративные российские сайты (чтобы никому не было обидно).

Адрес	Комментарий
	Англоязычные сайты
www.afralisp.com	Лучший, на наш взгляд, англоязычный сайт со множеством материалов по программированию с использованием Visual LISP и VBA
members.bellatlantic.net/~vze2vjds	**Delphi / AutoCAD Programming**. Сайт известного специалиста по САПР Тони Танзилло. Много интересных материалов, в том числе по программированию для AutoCAD с использованием Delphi
xarch.tu-graz.ac.at/autocad	**Xarch AutoCAD info + tools**. Авторитетнейший сайт Рейни Урбана и его единомышленников. Огромное количество материалов, главным отличием которых является красота LISP-функций
www.mcneel.com	Сайт, главным сокровищем на котором является известнейшая библиотека **DOSLib**
www.dsxcad.com	На этом сайте можно приобрести **The Visual LISP Developers Bible 2003 Edition** — очень полезную электронную книгу
www.manusoft.com/Software/Freebies/Main.stm	Множество полезных материалов, в том числе библиотека **AcadStatButton**, использованная нами
www.fleming-group.com	Сайт **Fleming Consulting Group** с библиотекой для работы из LISP с базами данных
www.cadwerx.net	Сайт с хорошим набором бесплатных инструментов, например **DwgPropsX.dll** в виде ActiveX control
www.dotsoft.com	Сайт, на котором находится известный комплект **ToolPack** и другие утилиты
www.jtbworld.com	Интересный сайт с хорошей подборкой материалов. Имеется перечень "багов" AutoCAD 2005

(продолжение)

Адрес	Комментарий
	Русскоязычные сайты
n.webring.com/hub?ring=russiancad	**Russian CAD-users Web Ring** (кольцо русскоязычных сайтов), поддерживаемое Виктором Ткаченко. На этом ресурсе можно найти ссылки на все русскоязычные САПР-ориентированные сайты, не упомянутые нами
www.cad.dp.ua	**Сайт пользователей САПР из exUSSR** Виктора Ткаченко. Один из самых насыщенных, полезных и популярных. На нем размещено много материалов как по системе AutoCAD, так и по другим системам. Интересен как пользователям, так и программистам
cadhlp.da.ru	**САПР Навигатор** — очень известный сайт Геннадия Поспелова. Множество материалов и бесплатных программ. Именно здесь развешивают известные многим российским пользователям "Довески"
www.autocad.ru/ cgi-bin/f1/board.cgi?&action=show_tree	Один из самых популярных форумов, на котором рассматриваются самые разнообразные темы, связанные с AutoCAD
www.autokad.ru	**AutoCAD для всех!** Обширный сайт Александра Острийчука. Масса полезных материалов, очень активный форум
www.cad.dp.ua/conf.php	**Web-конференция пользователей САПР-систем**, также поддерживаемая Виктором Ткаченко. К большому сожалению, после ряда проблем с бывшим хостингом, потеряла былую популярность
geol-dh.narod.ru	**Авторская страница Александра Косова**. Прикладные программы для геологов и геофизиков, горняков и маркшейдеров, проектировщиков, технологов, строителей, сантехников, гидротехников. Очень хороший сайт с хорошими программами
tyumsapr.narod.ru/cadsup	**САПР и Графика в Тюмени**. Имеется полезная документация по AutoCAD и программированию на русском языке
www.kulichki.com/libcad	Проект **Библиотека конструктора** Евгения Шувалова. Много полезных, но давно не обновлявшихся материалов
mirror01.iptelecom.net.ua/~olegkiev	**Проверено собой**. Ряд интересных статей по работе с AutoCAD. Поддерживает OLEg. Kiev

(окончание)

Адрес	Комментарий
www.arxmaster.by.ru	Сайт, посвященный программированию под AutoCAD с использованием ObjectARX. Поддерживает Иван Галака
www.alx.ncn.ru	**Plugins from Alex**. Сайт Александра Щетинина с прекрасной программой **ATable** и другими утилитами
aco.ifmo.ru/~nadinet	**Толстоба Н. Д.** Полезные материалы для изучения программирования в AutoCAD на начальном уровне. Предназначены для студентов, но могут оказаться полезными многим
engineer.rus.co.il	Израильский сайт **"Русскій" ИнженерЪ**, на котором имеются материалы по автоматизированному проектированию. Интересен тем, что позволяет оценить "их" интересы
www.cadburg.by.ru	**CADburg** — оренбургский сайт с неплохими статьями
www.dpi.dp.ua/software/acesoft.shtml	Страницы известного фаната LISP Сергея Попадьина. Интересные, хотя в чем-то и спорные статьи
www.cad.dp.ua/poleschuk.html, www.private.peterlink.ru/poleshchuk/cad	Сайты Николая Полещука
www.gis.kurgan.ru	**Геоинформационная система г. Кургана** — сайт Сергея Зуева, посвященный интеграции САПР и ГИС и предшественнице ruCAD — системе BestIA. Материалы сайта постепенно переезжают на сайт нашей книги

И наконец, приведем адрес сайта, на котором можно найти все дополнительные материалы нашей книги — **www.kurganobl.ru/cad**. На этом ресурсе можно ознакомиться со множеством материалов, не вошедших в нашу книгу, высказать мнение о ней, задать вопросы авторам, получить установочный комплект системы ruCAD и обновлений к ней.

Предметный указатель

H

I

L

M

N

O

P

R

S

T

U

V

X

А

Б

В

Г

Д

Е

З

И

К

Л

М

Н

О

П

Р

С

Т

У

Ф

Продолжение рубрики см. на с. 1160

Продолжение рубрики см. на с. 1162

Продолжение рубрики см. на с. 1164

Х

Ц

Ч

Ш

Э

Я