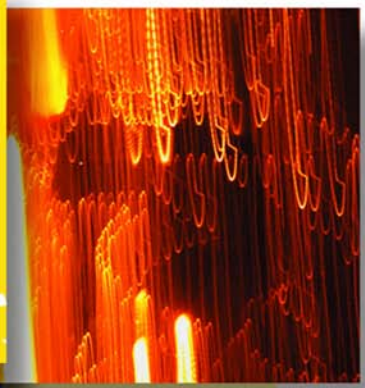
СЕРГЕЙ ПОПОВ

bhv®

# Delphi и 1С:Предприятие

## программирование информационного обмена



Основы конфигурирования системы 1С:Предприятие

Доступ к информационной структуре системы 1С:Предприятие с использованием механизма OLE Automation

Считывание и изменение информации с помощью SQL-запросов

Расширение возможностей и администрирование системы 1С:Предприятие

PRO
ПРОФЕССИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ

+ CD

Сергей Попов

# Delphi и 1С:Предприятие

## программирование информационного обмена

Санкт-Петербург

«БХВ-Петербург»

2007

УДК 681.3.06
ББК 32.973.26-018.1
П58

**Попов С. А.**

П58 Delphi и 1С:Предприятие. Программирование информационного обмена. — СПб.: БХВ-Петербург, 2007. — 592 с.: ил. + CD-ROM — (Профессиональное программирование)

ISBN 978-5-9775-0041-8

Рассмотрены вопросы, связанные с программированием информационного обмена между системой 1С:Предприятие и внешними приложениями, разработанными в Delphi. Приведены общие сведения о конфигурации системы 1С:Предприятие. На конкретных примерах показаны приемы программирования и доступ к информационной структуре системы 1С:Предприятие с использованием механизма OLE Automation, а также прямой доступ к данным с помощью SQL-запросов. Уделено внимание вопросам внешнего администрирования системы 1С:Предприятие. Материал сопровождается большим количеством практических примеров, размещенных на прилагаемом компакт-диске.

*Для программистов*

УДК 681.3.06
ББК 32.973.26-018.1

**Группа подготовки издания:**

| | |
|---|---|
| Главный редактор | *Екатерина Кондукова* |
| Зам. главного редактора | *Игорь Шишигин* |
| Зав. редакцией | *Григорий Добин* |
| Редактор | *Татьяна Темкина* |
| Компьютерная верстка | *Натальи Караваевой* |
| Корректор | *Наталия Першакова* |
| Дизайн серии | *Инны Тачиной* |
| Оформление обложки | *Елены Беляевой* |
| Зав. производством | *Николай Тверских* |

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 27.02.07.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 47,73.
Тираж 2000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.02.953.Д.006421.11.04 от 11.11.2004 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-9775-0041-8

© Попов С. А., 2007
© Оформление, издательство "БХВ-Петербург", 2007

# Оглавление

**ЧАСТЬ I. 1С И МЕХАНИЗМ OLE AUTOMATION ..... 1**

**Глава 1. Общие сведения ..... 3**

**Глава 2. Подготовка к изучению материала этой части. Конфигурация 1С ..... 4**

**Глава 3. Подключение к 1С ..... 19**

3.1. Первый этап ..... 19
3.2. Второй этап ..... 25

**Глава 4. Доступ к информации 1С из внешних приложений ..... 29**

4.1. Считывание информации из справочников ..... 31
4.1.1. Реквизит справочника типа *Перечисление* ..... 32
4.1.2. Реквизит справочника типа *Строка неограниченной длины* ..... 42
4.1.3. Реквизит справочника типа *Справочник* ..... 51
4.1.4. Обобщение пройденного материала ..... 62
4.1.5. Периодические реквизиты справочников ..... 86
4.2. Считывание информации из документов ..... 123
4.3. Считывание информации из регистров ..... 157
4.4. Доступ к информационной структуре базы данных 1С ..... 187
4.4.1. Доступ к информационной структуре с использованием механизма OLE Automation ..... 187
4.4.2. Doc-файлы (хранилища структурированной информации). Файл конфигурации системы 1С:Предприятие — 1Cv7.MD ..... 215
4.5. Метод *EvalExpr* ..... 228
4.6. Метод *ExecuteBatch* ..... 243
4.7. Изменение информации в базе данных 1С ..... 266
4.7.1. Редактирование элементов справочников ..... 266
4.7.2. Редактирование документов ..... 333
4.8. Синхронизация баз данных 1С ..... 342

**Глава 5. Расширение функционала 1С ..... 359**

5.1. Общие сведения ..... 360
5.2. Внутрипроцессный сервер автоматизации ..... 362
5.3. Внепроцессный сервер автоматизации ..... 433

**ЧАСТЬ II. ДОСТУП К БАЗЕ ДАННЫХ 1С БЕЗ ПРОГРАММЫ 1С:ПРЕДПРИЯТИЕ ..... 451**

**Глава 6. Файлы формата DBF ..... 453**

6.1. Общая информация о DBF-формате ..... 453
6.2. Доступ к информации в DBF-файлах ..... 456

**Глава 7. Обработка словаря базы данных 1С (файл 1Cv7.DD) ..... 462**

**Глава 8. Доступ к таблицам базы данных с учетом информации в файле 1Cv7.DD ..... 468**

**Глава 9. Дополнительные данные об информационном содержании таблиц, полученные с помощью механизма OLE Automation ..... 479**

**Глава 10. Дополнительные данные об информационном содержании таблиц, полученные из файла 1Cv7.MD ..... 483**

**Глава 11. Остатки ТМЦ на складах. Пример ..... 526**

**ЧАСТЬ III. ПРОГРАММА 1С:ПРЕДПРИЯТИЕ В СРЕДЕ WINDOWS ..... 551**

**Глава 12. Список зарегистрированных баз даннных системы 1С:Предприятие ..... 553**

**Глава 13. Список активных пользователей базы данных 1С:Предприятие ..... 558**

**Глава 14. Отслеживание регламента системного обслуживания баз данных 1С:Предприятие ..... 566**

**Глава 15. Программа 1С:Предприятие в качестве DDE-сервера ..... 570**

**Приложение. Описание компакт-диска ..... 573**

Структура содержимого компакт-диска ..... 573

**Предметный указатель ..... 575**

## Часть I

# 1С и механизм OLE Automation

Эта часть посвящена вопросам программирования информационного обмена между системой 1С и внешними приложениями с применением механизма OLE Automation.

# Глава 1

# Общие сведения

OLE Automation представляет собой механизм управления приложением извне с помощью соответствующих сервисов, предоставляемых самим приложением. Для реализации этих сервисов приложение использует интерфейсы COM-объектов, которые располагаются внутри его адресного пространства. Такое приложение называется *сервером OLE Automation*, или *сервером автоматизации*.

Приложение, использующее эти сервисы, называется *контроллером автоматизации*, или *клиентом автоматизации*. Сервер автоматизации может выполняться как в собственном адресном пространстве (в виде приложения), так и в адресном пространстве клиента (в виде DLL-библиотеки). В первом случае сервер называется *внешним* (out-of-process — внепроцессный), а во втором случае — *встроенным* (in-process — внутрипроцессный).

Любой сервер OLE Automation, а также COM-объект, идентифицируется уникальным 128-битовым (16-байтным) кодом GUID (Global Unique Identifier — глобально уникальный идентификатор). Информация обо всех серверах OLE Automation и COM-объектах хранится в системном реестре. Сервисные возможности сервера автоматизации определяются объектами, методами и свойствами, которые разработчики сервера предусмотрели для использования во внешних приложениях.

Если сервер и клиент выполняются на разных компьютерах, то сервер называется *удаленным*. В рамках данной книги такие серверы рассматриваться не будут.

В качестве примера серверов OLE Automation можно привести такие приложения, как Microsoft Office (Word, Excel и т. д.), Microsoft Internet Explorer и др. Система 1С также поддерживает механизм OLE Automation в качестве как сервера, так и клиента. Следовательно, здесь применимы все приемы и способы программирования, разработанные для использования этой технологии в других приложениях.

# Глава 2

# Подготовка к изучению материала этой части. Конфигурация 1С

Те, у кого есть опыт работы в среде Конфигуратора системы 1С:Предприятие, могут пропустить эту главу.

Для того чтобы можно было уделить максимум внимания процессу программирования, не отвлекаясь на нюансы конфигураций 1С, доступные читателям, создадим свою, простую конфигурацию 1С.

> ***Примечание***
>
> Создаваемая конфигурация носит отвлеченный от реалий характер и не претендует на какую-либо практическую ценность для ее использования в иных целях, кроме учебных.

> ***Примечание***
>
> Конфигурация размещена на сопроводительном компакт-диске книги в папке \1C_example.DB\.

Рассмотрим весь процесс создания конфигурации 1С достаточно подробно, чтобы читатели, не имеющие опыта работы с 1С, получили представление об этом процессе.

В данном случае будет создана локальная конфигурация на основе файлов формата DBF. Для этого необходимо создать папку на каком-либо логическом диске компьютера (например, E:\1C_example.DB\) и запустить Конфигуратор системы 1С:Предприятие, выбрав его в программном меню Windows (рис. 2.1) или щелкнув на его ярлыке (рис. 2.2).

На экране появится диалоговое окно со списком конфигураций 1С (рис. 2.3).

Здесь (в данном случае) нужно нажать кнопку **Добавить**. На экране появится диалоговое окно (рис. 2.4), где следует ввести наименование конфигурации и папку, где будут размещены файлы конфигурации и базы данных 1С.

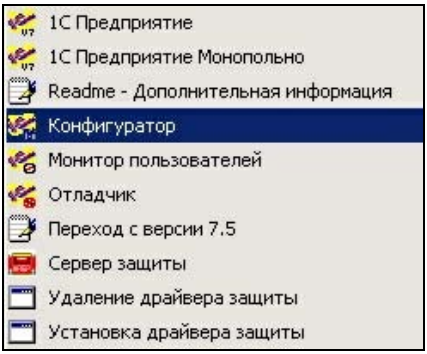



**Рис. 2.1.** Запуск конфигуратора из программного меню Windows

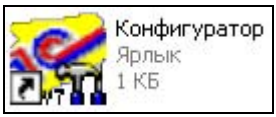



**Рис. 2.2.** Ярлык конфигуратора

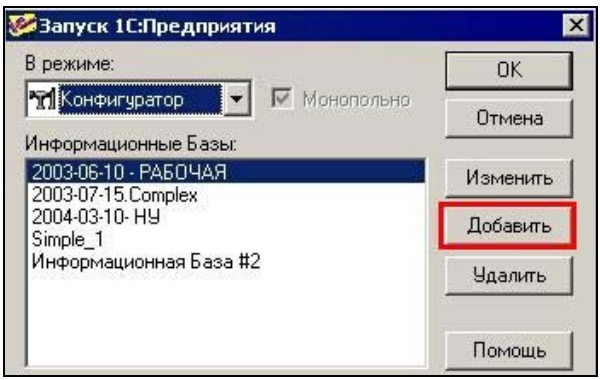



**Рис. 2.3.** Диалоговое окно выбора базы данных 1С

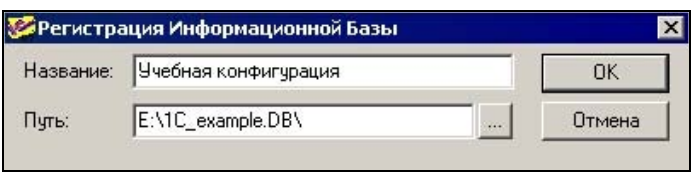



**Рис. 2.4.** Диалоговое окно ввода информации о новой конфигурации

После этого новая конфигурация появится в списке конфигураций (рис. 2.5). Следует отметить, что при обращении к 1С через механизм OLE Automation это диалоговое окно появляется только в том случае, если запрашиваемая конфигурация не найдена.

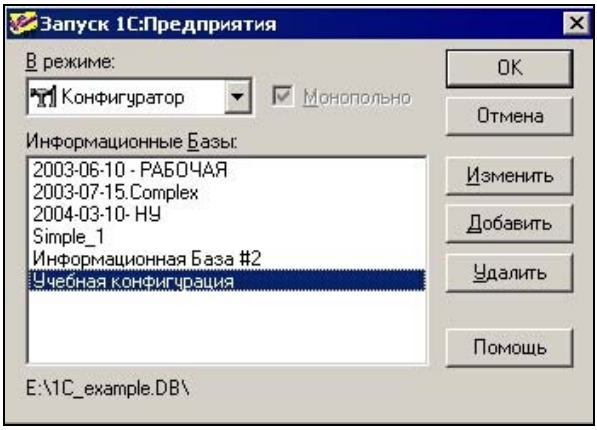



**Рис. 2.5.** Диалоговое окно выбора базы данных 1С (появилась новая конфигурация)

Чтобы можно было настроить (реорганизовать) конфигурацию, следует выбрать ее в списке (см. рис. 2.5) и нажать кнопку **OK**.

Если конфигурация новая, на экране появится диалоговое окно для выбора типа базы данных: локальная (на основе DBF-файлов) или удаленная ("клиент-сервер") в среде MS SQL Server (рис. 2.6).

В нашем случае необходимо указать вариант локальной базы данных: **Файлы *.DBF; *.CDX**, а также формат кодировки кириллицы (**1251**).

На рис. 2.7 показан список файлов и папок, созданных программой 1С для пустой конфигурации.

***Примечание***

Информационная модель конфигураций 1С в недостаточной степени документирована. И это представляет трудности в случаях, когда требуется обеспечить доступ к данным из других приложений, вне контекста системы 1С:Предприятие. Эти вопросы будут более подробно рассмотрены в *части II*.

Чтобы изменить конфигурацию (реорганизовать базу данных), необходимо ее открыть. Это можно сделать из меню Конфигуратора или нажав на соответствующую кнопку (рис. 2.8).

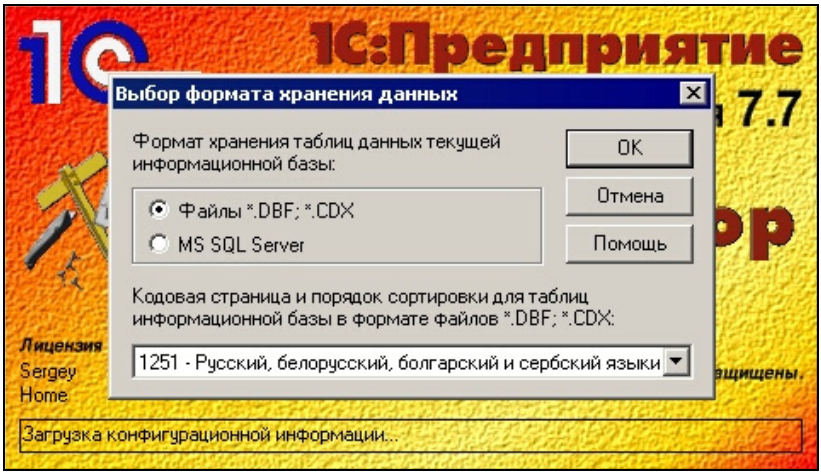



**Рис. 2.6.** Диалоговое окно выбора типа базы данных 1С

| Имя | Размер | Тип | Изменен |
|---|---|---|---|
| SYSLOG | | Папка с файлами | 12.06.2006 14:27 |
| 1Cv7.LCK | 0 КБ | Файл "LCK" | 12.06.2006 14:27 |

**Рис. 2.7.** Список файлов в папке E:\1C_example.DB\ (пустая конфигурация)

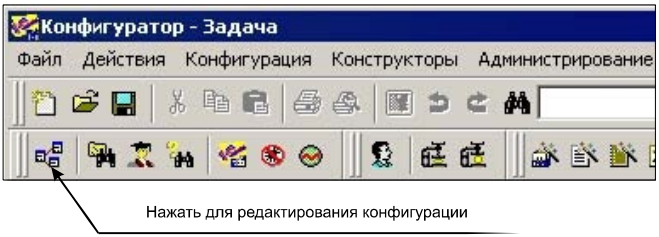



**Рис. 2.8.** Панель инструментов конфигуратора 1С (кнопка редактирования)

Создадим простой справочник "Список физических лиц".

### *Примечание*

В терминологии 1С *справочником* называется объект программы, позволяющий пользователю вводить, хранить и получать информацию, структурируя ее в виде дерева. Справочник представляется списком древовидной структуры, в узлах которого хранится информация о различных объектах. Информация хранится в виде записей, все узлы дерева хранят записи одной структуры, содержащей различные величины. Набор этих величин для каждого справочника произволен за исключением двух строковых величин: кода объекта и значе-

ния объекта. *Код объекта* является уникальным для данного справочника и позволяет ссылаться на этот объект из других мест программы. *Значение объекта* — произвольная строка, введенная пользователем (обычно это название объекта).

Для создания нового справочника следует в конфигурации выделить строку **Справочники** и, вызвав всплывающее меню, выбрать пункт **Новый справочник** (рис. 2.9).

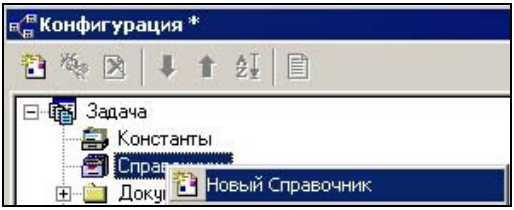



**Рис. 2.9.** Всплывающее меню конфигуратора

На экране появится диалоговое окно настройки справочника (рис. 2.10).

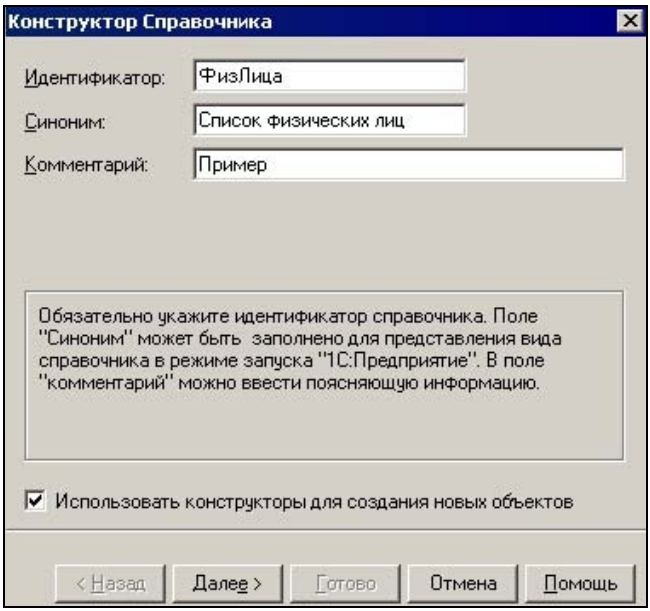



**Рис. 2.10.** Диалоговое окно создания нового справочника (идентификатор справочника)

Идентификатор справочника должен быть *уникальным* в контексте данной конфигурации.

> ***Примечание***
>
> В терминологии 1С *идентификатор* — краткое наименование объекта метаданных. Идентификатор используется для обозначения объекта метаданных в списках и диалогах, а также служит для обращения к объекту метаданных в программных модулях.
>
> Все обращения через механизм OLE Automation к этому справочнику также производятся с использованием этого идентификатора.

Чтобы продолжить создание справочника, нажмите кнопку **Далее** (рис. 2.10). На экране поочередно появятся диалоговые окна (рис. 2.11 и 2.12). Здесь также нужно нажать кнопки **Далее** и **Готово** соответственно, ничего не меняя.

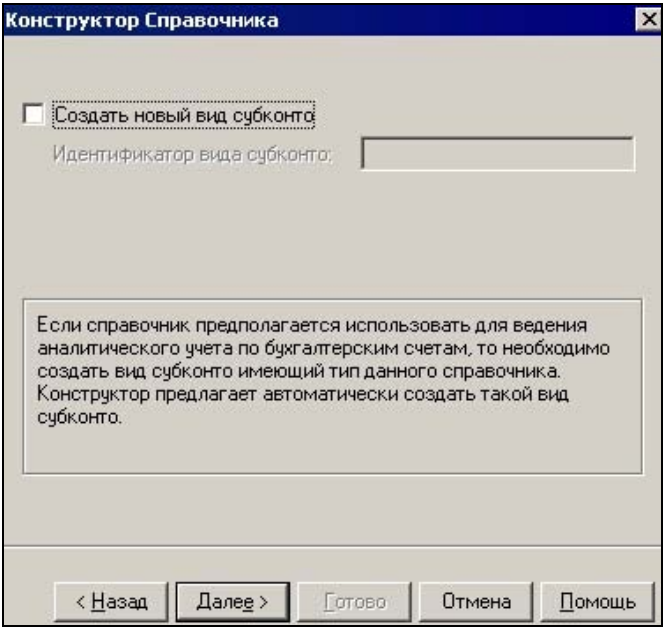



**Рис. 2.11.** Диалоговое окно создания нового справочника (продолжение)

На рис. 2.13 представлено диалоговое окно настройки справочника. Создаваемый справочник прост, потому что предназначен для тренировочных целей. Значения всех полей, кроме **Длина наименования**, можно оставить установленными по умолчанию.

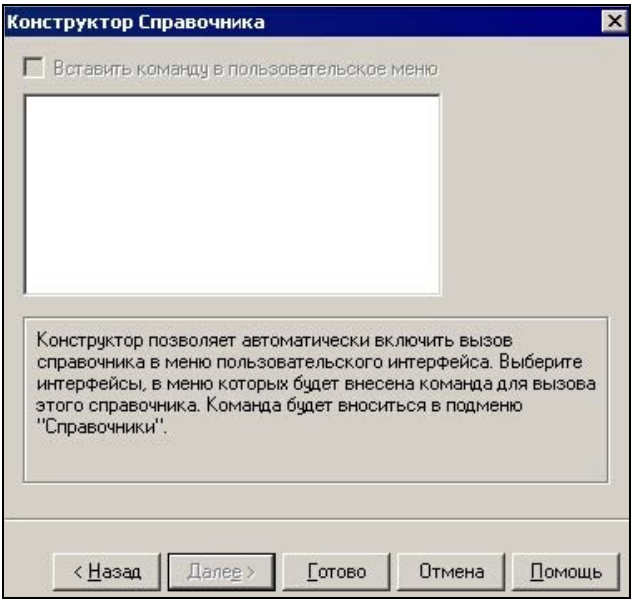



**Рис. 2.12.** Диалоговое окно создания нового справочника (продолжение)

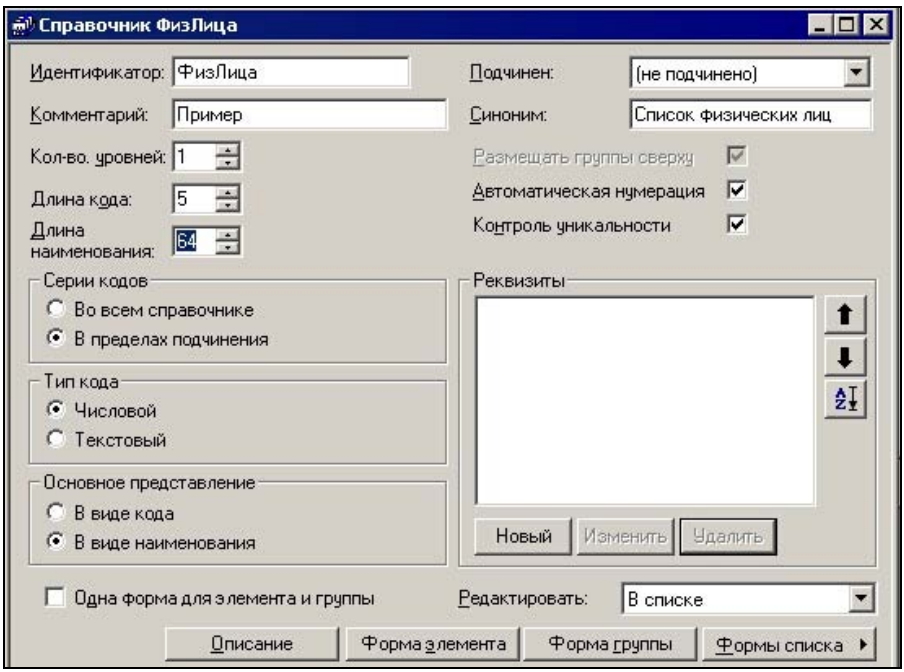



**Рис. 2.13.** Диалоговое окно создания нового справочника (свойства и реквизиты справочника)

Поле **Длина наименования** представляет интерес потому, что в свойстве `Наименование` будут храниться фамилия, имя и отчество физического лица. Поэтому необходимо увеличить значение его длины относительно принятого по умолчанию. Здесь следует отметить, что код и наименование являются общими свойствами всех справочников конфигурации 1С. Максимальная длина кода элемента справочника — 24 символа.

Если в справочнике нужно хранить дополнительную информацию, то для этого используются реквизиты справочника. Расширим справочник дополнительной информацией:

- Дата рождения (тип `Дата`);
- Домашний адрес (тип `Строка`).

Чтобы внести новый реквизит в справочник, нужно нажать кнопку **Новый** в группе **Реквизиты** (см. рис. 2.13). Затем в диалоговом окне настройки реквизита (рис. 2.14 и 2.15) следует указать идентификатор реквизита, его синоним и тип. В контексте данного справочника идентификатор реквизита должен быть уникальным. В зависимости от типа реквизита может потребоваться внести дополнительную информацию о реквизите, например: `Длина` (для типа `Строка` или `Число`), `Точность` (для типа `Число`) и другие данные (на вкладке **Дополнительные**).

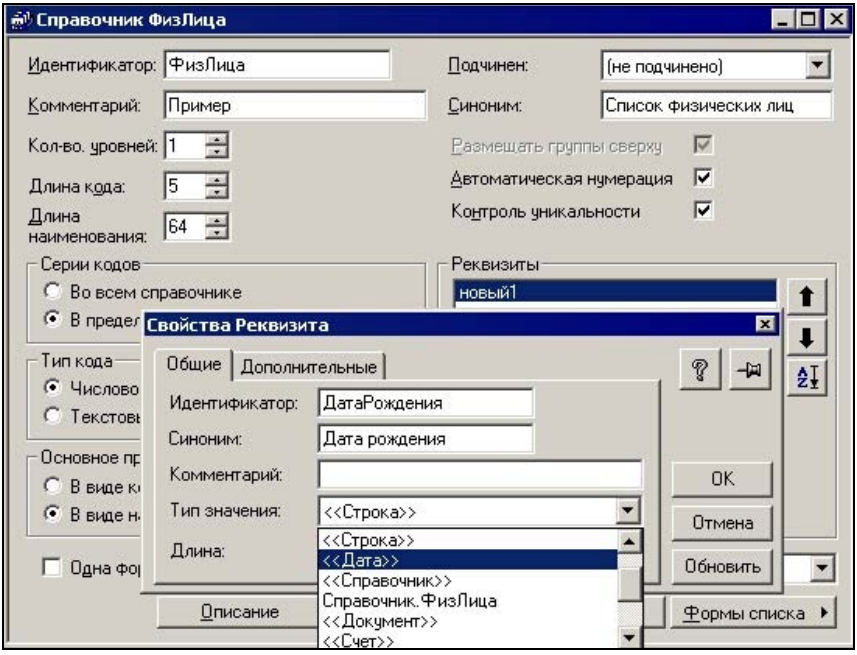



**Рис. 2.14.** Диалоговое окно создания нового справочника (новый реквизит типа `Дата`)

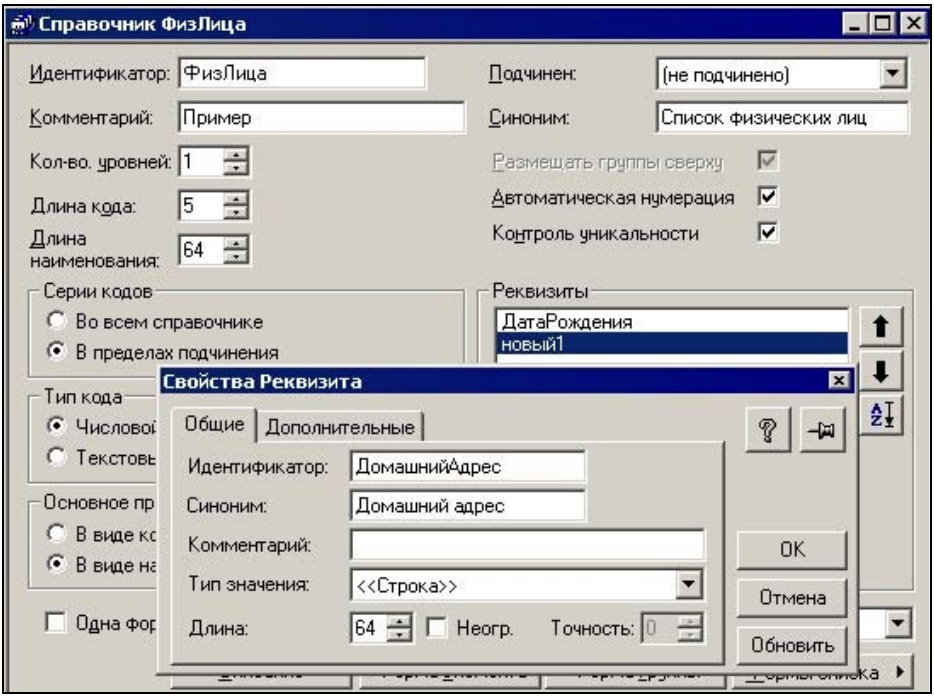



**Рис. 2.15.** Диалоговое окно создания нового справочника (новый реквизит типа `Строка`)

В задачу данной книги не входит детальное описание возможностей 1С, поэтому ограничимся простым справочником.

На рис. 2.16 представлено дерево конфигурации, где отражен новый справочник.

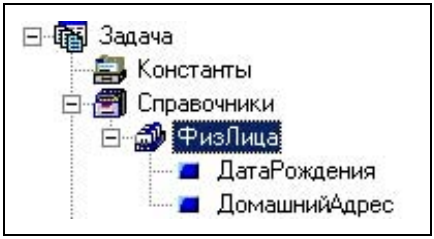



**Рис. 2.16.** Новый справочник в списке конфигурации 1С

Чтобы сохранить сделанные в конфигурации изменения, нужно нажать соответствующую кнопку на панели Конфигуратора 1С (рис. 2.17).

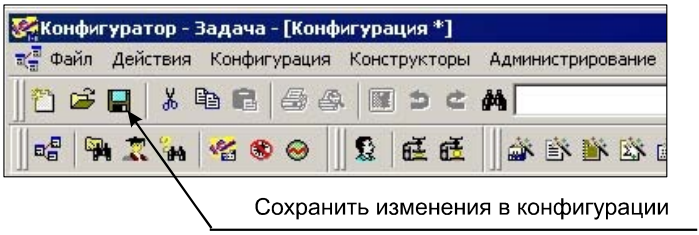



**Рис. 2.17.** Панель инструментов конфигуратора 1С (кнопка сохранения конфигурации)

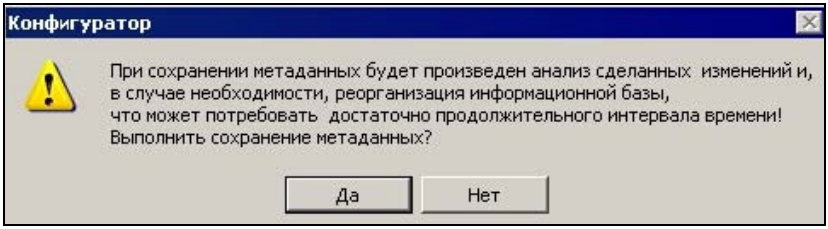



**Рис. 2.18.** Запрос на подтверждение сохранения изменений в конфигурации

Далее следует запрос на подтверждение (рис. 2.18).

На рис. 2.19 представлен запрос, характерный для новой конфигурации или для случаев, когда по каким-то причинам отсутствует файл 1Cv7.DD.

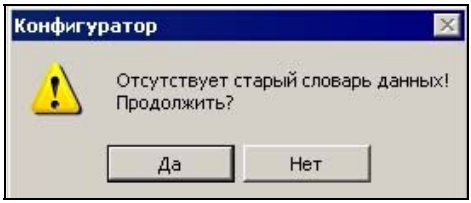



**Рис. 2.19.** Запрос на создание файла 1Cv7.DD

Здесь следует отметить, что текстовый файл 1Cv7.DD — это словарь данных. Имя этого файла зависит от формата хранения таблиц базы данных (см. рис. 2.6). Для локальной базы данных (формат DBF) имя файла: 1Cv7.DD. В случае, если база данных размещается в среде MS SQL Server, имя этого файла: 1Cv7.DDS.

***Примечание***

Более детально структура словаря данных (1Cv7.DD) будет рассмотрена в *части II*.

Далее Конфигуратор представит список сделанных изменений. Если эти изменения корректны, их следует принять, нажав соответствующую кнопку (рис. 2.20).

После реорганизации структуры на экран выводится соответствующее сообщение (рис. 2.21).

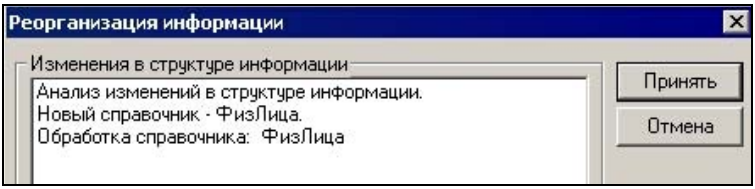



**Рис. 2.20.** Список изменений в конфигурации, которые будут сохранены

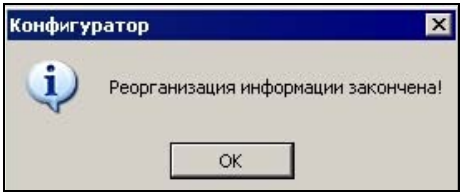



**Рис. 2.21.** Сообщение об окончании реорганизации конфигурации

| Имя | Размер | Тип | Изменен |
|---|---|---|---|
| SYSLOG | | Папка с файлами | 12.06.2006 14:27 |
| 1Cv7.LCK | 0 КБ | Файл "LCK" | 12.06.2006 14:27 |
| NEW_STRU | | Папка с файлами | 12.06.2006 15:06 |
| 1Cv7.DD | 9 КБ | Файл "DD" | 12.06.2006 15:06 |
| 1SBLOB | 5 КБ | Active Server Docu... | 12.06.2006 15:07 |
| 1SBLOB.DBF | 1 КБ | Файл "DBF" | 12.06.2006 15:07 |
| 1SCONST | 6 КБ | Active Server Docu... | 12.06.2006 15:07 |
| 1SCONST.DBF | 1 КБ | Файл "DBF" | 12.06.2006 15:07 |
| 1SCRDOC | 6 КБ | Active Server Docu... | 12.06.2006 15:07 |
| 1SCRDOC.DBF | 1 КБ | Файл "DBF" | 12.06.2006 15:07 |
| 1SDNLOCK | 5 КБ | Active Server Docu... | 12.06.2006 15:07 |
| 1SDNLOCK.DBF | 1 КБ | Файл "DBF" | 12.06.2006 15:07 |
| 1SJOURN | 11 КБ | Active Server Docu... | 12.06.2006 15:07 |
| 1SJOURN.DBF | 1 КБ | Файл "DBF" | 12.06.2006 15:07 |
| 1SSYSTEM.DBF | 1 КБ | Файл "DBF" | 12.06.2006 15:07 |
| 1SUIDCTL | 5 КБ | Active Server Docu... | 12.06.2006 15:07 |
| 1SUIDCTL.DBF | 1 КБ | Файл "DBF" | 12.06.2006 15:07 |
| 1SUSERS.DBF | 1 КБ | Файл "DBF" | 12.06.2006 15:07 |
| 1Cv7.MD | 63 КБ | Файл "MD" | 12.06.2006 15:07 |
| SC12 | 8 КБ | Active Server Docu... | 12.06.2006 15:07 |
| SC12.DBF | 1 КБ | Файл "DBF" | 12.06.2006 15:07 |

**Рис. 2.22.** Список файлов в папке E:\1C_example.DB\ (после реорганизации)

На этом простом примере рассмотрение вопроса создания и изменения конфигурации 1С:Предприятие заканчивается. Те, кто желает ознакомиться с этим процессом более детально, должны использовать соответствующую специальную литературу.

В дальнейшем, по ходу изложения, мы будем неоднократно наращивать конфигурацию, дополняя ее другими объектами метаданных.

Определенный интерес для программиста может представлять содержание папки, где размещена созданная конфигурация (E:\1C_example.DB\).

На рис. 2.22 показан список файлов и папок, созданных программой 1С после того, как был создан справочник "Список физических лиц".

Назначение некоторых файлов приведено в табл. 2.1.

***Таблица 2.1.*** *Файлы в папке конфигурации системы 1С:Предприятие*

| **Файл** | **Назначение** |
|---|---|
| 1Cv7.MD | Файл конфигурации. Здесь хранится вся информационная структура данных. Редактирование этого файла осуществляется Конфигуратором 1С |
| 1CV7.DD | Словарь данных (текстовый файл). Содержит описания структуры таблиц и индексов базы данных программы 1С:Предприятие. Этот файл генерируется автоматически |
| 1Cv7.LCK | Появляется в том случае, если конфигурация открыта программой 1С:Предприятие или Конфигуратором |
| 1SBLOB.* | Здесь хранятся значения реквизитов справочников, документов и счетов имеющих тип `Строка неограниченной длины` |
| 1SCONST.* | Здесь хранятся значения констант, периодических реквизитов справочников и бухгалтерских счетов |
| 1SCRDOC.* | Здесь хранятся вхождения документов в графы отбора, списки подчиненных документов, вхождения документов в общие журналы, для которых определен состав документов |
| 1SDNLOCK.* | Здесь хранится временный список номеров документов, которые вводятся в данный момент |
| 1SJOURN.* | Здесь хранятся журналы документов |
| 1SSYSTEM.DBF | Содержит общие параметры информационной базы (точка актуальности, рассчитанный период бухгалтерских итогов, периодичность оперативных итогов и т. п.) |
| 1SUIDCTL.* | Здесь хранятся уникальности (дополнительный контроль уникальности внутренней идентификации объектов) |
| 1SUSERS.DBF | Содержит число соединений с базой данных, счетчик изменений данных пользователями |
| SC12.* | Справочник "Список физических лиц" (в нашем случае) |

Чтобы заполнить созданный справочник "Список физических лиц" информацией, нужно запустить программу 1С:Предприятие. Сделать это можно из Конфигуратора, нажав соответствующую кнопку (рис. 2.23).

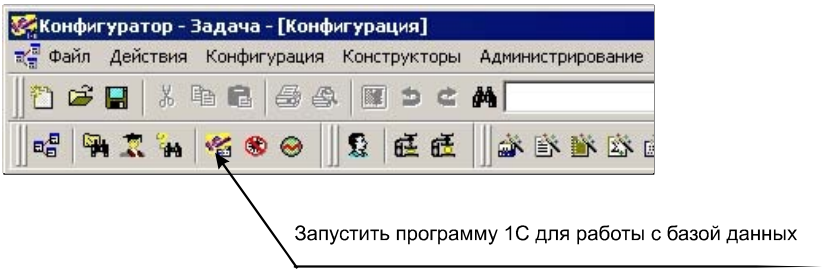



**Рис. 2.23.** Панель инструментов конфигуратора 1С (кнопка запуска 1С:Предприятия)

Когда программа 1С:Предприятие запустится, откроем список доступных справочников с помощью меню (рис. 2.24).

В списке доступных справочников выберем строку **Список физических лиц (Пример)** и нажмем кнопку **OK** (рис. 2.25).

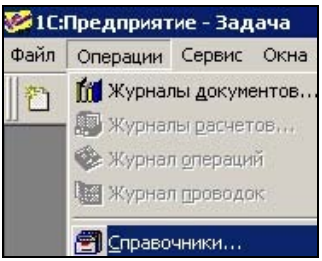



**Рис. 2.24.** Меню 1С:Предприятие (выбор списка справочников для редактирования)

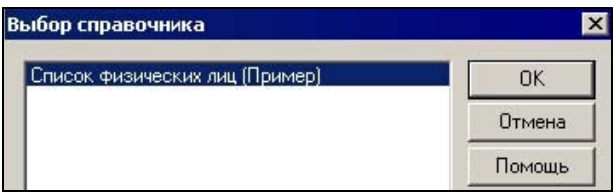



**Рис. 2.25.** Список справочников для выбора

На экране появится окно, отражающее содержание справочника. В данном случае справочник пуст. Чтобы внести информацию, вызовем всплывающее меню и выберем пункт **Новый** (рис. 2.26).

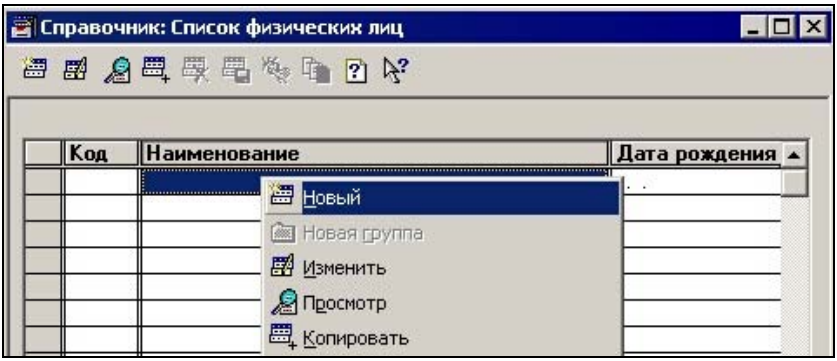



**Рис. 2.26.** Всплывающее меню для работы со справочником

Затем внесем фамилию, имя, отчество, домашний адрес и дату рождения для физического лица, как показано на рис. 2.27.

Повторяя эту процедуру, внесем информацию о других физических лицах.

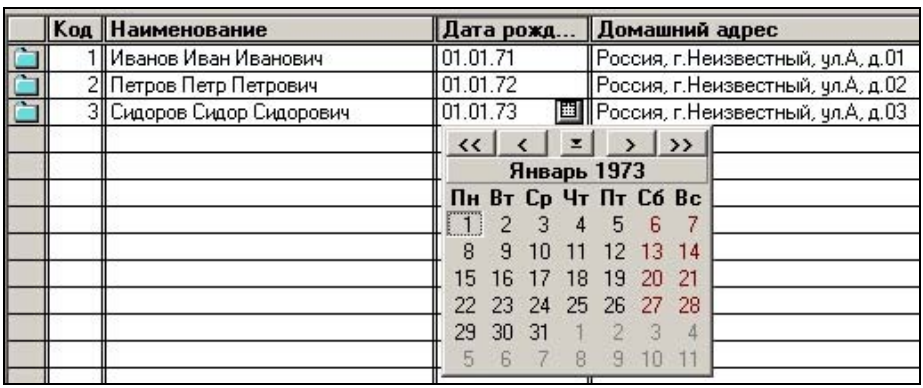



**Рис. 2.27.** Ввод данных в справочник `ФизЛица`

На рис. 2.28 представлен список введенных физических лиц.

Как видим, список отсортирован не в том порядке, как данные вводились. Следует отметить, что в данном случае порядок сортировки был определен выбором переключателя в группе **Основное представление** (см. рис. 2.13). По умолчанию в этой группе выбран переключатель **В виде наименования**, что и определило порядок сортировки.

Созданный нами справочник отражается в словаре данных (файл 1Cv7.DD) так, как показано на рис. 2.29.

| Код | Наименование | Дата... | Домашний адрес |
|---|---|---|---|
| 5 | Аксакалов Аксакал Аксакалович | 01.01.70 | Россия, г.Б, ул.Б, д.00 |
| 1 | Иванов Иван Иванович | 01.01.71 | Россия, г.Неизвестный, ул.А, д.01 |
| 2 | Петров Петр Петрович | 01.01.72 | Россия, г.Неизвестный, ул.А, д.02 |
| 3 | Сидоров Сидор Сидорович | 01.01.73 | Россия, г.Неизвестный, ул.А, д.03 |
| 4 | Федоров Федор Федорович | 01.01.74 | Россия, г.Известный, ул.Б., д.001 |

**Рис. 2.28.** Содержание справочника `ФизЛица` после редактирования (сортировка по наименованию)

```
#==============================================================================
#==TABLE no 8       : Справочник ФизЛица
# Name     |Descr                          |Type[A/S/U]|DBTableName|ReUsable
T=SC12     |Справочник ФизЛица             |A          |SC12       |1
#-----Fields-------
# Name       |Descr                |Type|Length|Precision
F=ID         |ID object            |C   |9     |0
F=CODE       |object code          |C   |5     |0
F=DESCR      |object description   |C   |64    |0
F=ISMARK     |Flag Object is Marke|C   |1     |0
F=VERSTAMP   |Version stamp        |C   |6     |0
F=SP15       |(P)ДатаРождения      |D   |8     |0
F=SP16       |(P)ДомашнийАдрес     |C   |64    |0
#----Indexes------
# Name      |Descr          |Unique|Indexed fields                                  |DBName
I=IDD       |of ID          |0     |ID                                              |IDD
I=CODE      |of CODE        |0     |CODE(UPPER)                                     |CODE
I=DESCR     |of DESCR       |0     |DESCR(UPPER)                                    |DESCR
```

**Рис. 2.29.** Содержание файла 1Cv7.DD

Эта информация будет полезна при рассмотрении вопросов доступа к базе данных 1С без использования механизма OLE Automation.

Для того чтобы подробнее изучить доступ к информации базы данных 1С, в дальнейшем нужно будет расширить созданную конфигурацию. Мы будем делать это по ходу изложения материала.

В следующей главе рассматриваются вопросы подключения к системе 1С:Предприятие как к серверу OLE Automation.

# Глава 3

# Подключение к 1С

Для иллюстрации механизма подключения к программе 1С:Предприятие создадим в среде Delphi простое приложение, которое будет считывать информацию из справочника "Список физических лиц", созданного в предыдущей главе.

## 3.1. Первый этап

***Примечание***

Исходные тексты содержатся на сопроводительном компакт-диске книги в файле Part_1\Chapter_03\Topic_3_1\sp_Connect_to_1C_01.PAS.

1. Создадим новый проект.
2. Поместим в форму два компонента `TPanel` (свойство `Align` компонента `Panel1` установим равным `AlTop`, а компонента `Panel2` — `AlClient`).
3. На панель `Panel1` добавим кнопку `Button1` (`TButton`) и установим для нее свойство `Caption = "Соединить"`.
4. Добавим на панель `Panel1` еще 4 компонента — `Label1`, `Edit1`, `Label2`, `Edit2` — и установим их свойства:
   - `Label1.Caption = "Имя 1С-сервера = "`
   - `Label2.Caption = "Строка инициализации = "`
   - `Edit1.Text = "V77.Application"`
   - `Edit2.Text = " /DE:\1C_example.DB "`
5. Поместим на панель `Panel2` компонент `ListBox1` (`TListBox`); его свойство `Align` установим равным `AlClient`.

   Внешний вид созданной формы представлен на рис. 3.1.

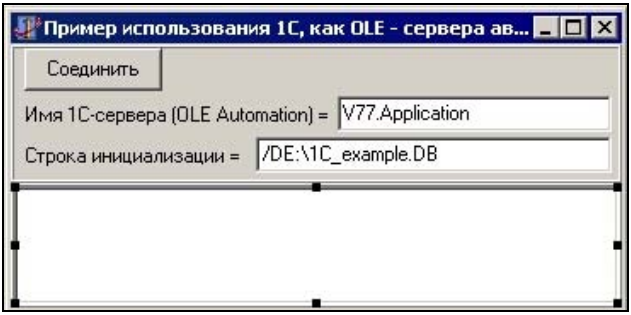



**Рис. 3.1.** Форма примера подключения к 1С (первый этап)

Поле `Edit1.Text` содержит имя 1С как сервера OLE Automation (мы указали `V77.Application`). Возможные значения (согласно документации по 1С):

- `V1CEnterprise.Application` — независимый ключ;
- `77.Application` — зависимый ключ;
- `77S.Application` — зависимый ключ, SQL-версия;
- `77L.Application` — зависимый ключ, локальная версия;
- `77M.Application` — зависимый ключ, сетевая версия.

Поле `Edit2.Text` содержит строку инициализации 1С. Мы указали `/DE:\1C_example.DB`, т. е. местонахождение базы данных, к которой желаем подключиться.

Некоторые возможные значения ключей (согласно документации по 1С):

`/D` *`Папка`* — имя каталога (папки), где располагается база данных;

`/M` — монопольный доступ;

`/N` *`Имя пользователя`*;

`/P`*`Пароль доступа`*.

Пример: `/DE:\1C_example.DB /NАдмин /P123`.

6. Дополним исходный текст программы (в листинге 3.1 изменения выделены полужирным).

**Листинг 3.1. Изменения в исходном тексте программы**

```
uses
ComObj,
Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
Dialogs, ExtCtrls, StdCtrls;


type
```

```
TForm1 = class(TForm)
    Panel1: TPanel;
    Panel2: TPanel;
    Button1: TButton;
    Label1: TLabel;
    Label2: TLabel;
    Edit1: TEdit;
    Edit2: TEdit;
    ListBox1: TListBox;
    procedure Button1Click(Sender: TObject);
private
    { Private declarations }
    Ole1C: OleVariant;
public
    { Public declarations }
end;
```

Переменная `Ole1C` (типа `OleVariant`) будет использована в дальнейшем, как указатель на объект 1С (сервер OLE Automation).

7. Сделаем двойной щелчок на компоненте `Button1` и изменим стандартный обработчик события (нажатия кнопки), как показано в листинге 3.2.

**Листинг 3.2. Вызов и инициализация программы 1С:Предприятие (сервер OLE Automation)**

```
procedure TForm1.Button1Click(Sender: TObject);
Var
  ResOpen: integer;
begin
  Ole1C:= CreateOleObject(Edit1.Text);
  ResOpen:=Ole1C.Initialize(Ole1C.RMTrade, Edit2.Text, '');
end;
```

Здесь следует сделать некоторые пояснения.

Строка

```
Ole1C:= CreateOleObject(Edit1.Text);
```

собственно, "запускает" программу 1С как OLE-сервер автоматизации (имя указано в `Edit1.Text`).

Строка

```
ResOpen:=Ole1C.Initialize(Ole1C.RMTrade, Edit2.Text, '');
```

инициализирует программу 1С согласно информации, содержащейся в `Edit2.Text`.

***Примечание***

Подробную информацию можно найти в соответствующей документации по 1С.

8. Сохраним проект под именем sp_Connect_to_1C_01.DPR (или любым другим, на усмотрение читателя), откомпилируем его и запустим приложение sp_Connect_to_1C_01.EXE.
9. Нажмем кнопку **Соединить**. Если все указано корректно (поля `Edit1.Text` и `Edit2.Text`), то программа 1С запустится как сервер OLE Automation.

Если на компьютере не установлена программа 1С:Предприятие, появится сообщение (рис. 3.2).

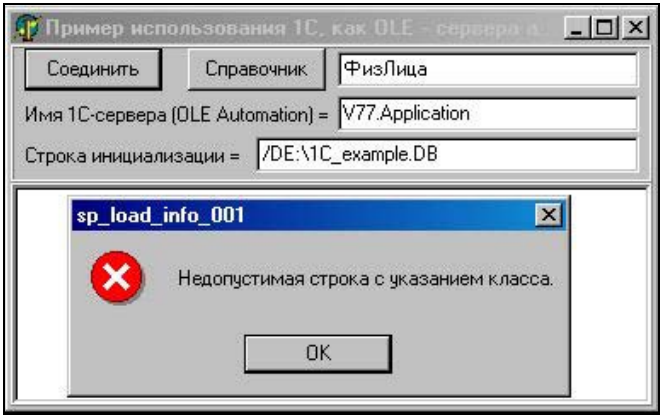



**Рис. 3.2.** Сообщение, появляющееся в случае, если на данном компьютере программа 1С не установлена

На рис. 3.3 показан внешний вид окна программы 1С:Предприятие в момент загрузки конфигурации.

Если в строке инициализации не указаны имя пользователя и (или) пароль доступа, то может последовать запрос на авторизацию (рис. 3.4).

Если некорректно указан каталог конфигурации (базы данных), то появится соответствующий запрос (рис. 3.5).

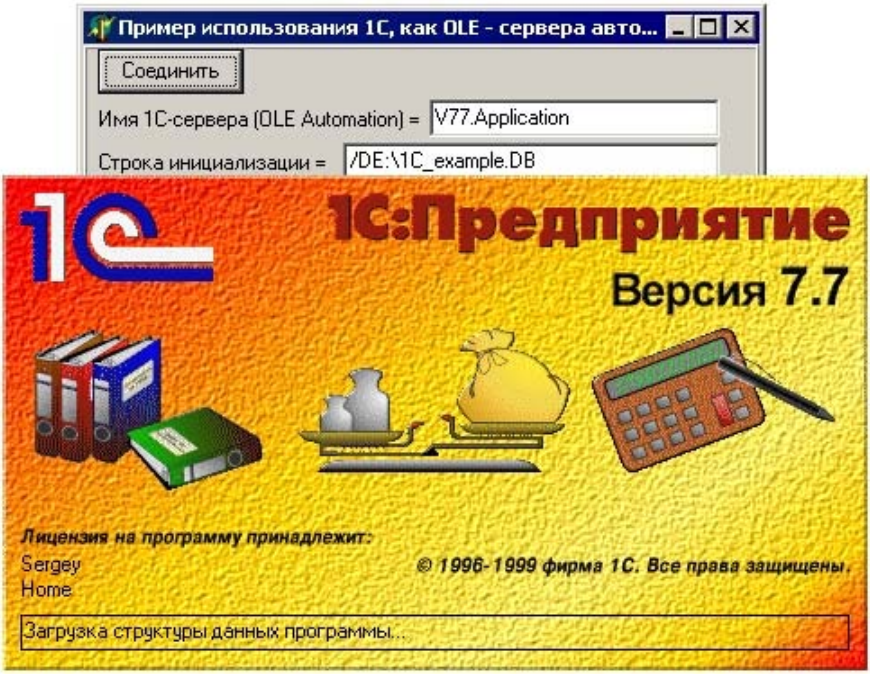



**Рис. 3.3.** Программа 1С:Предприятие запустилась, как сервер OLE Automation

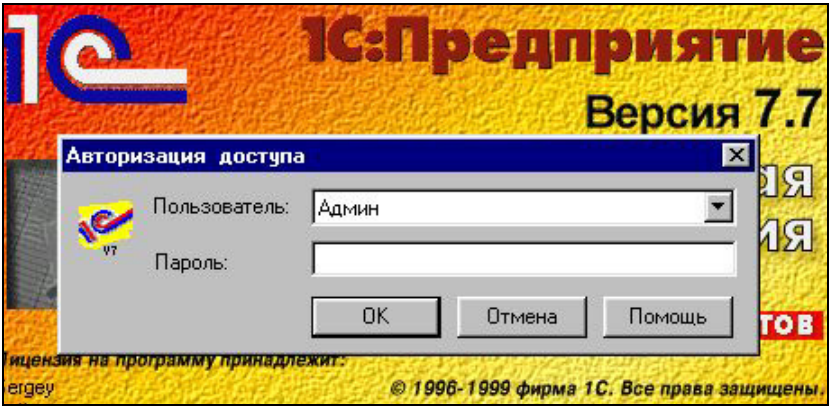



**Рис. 3.4.** Запрос на авторизацию пользователя

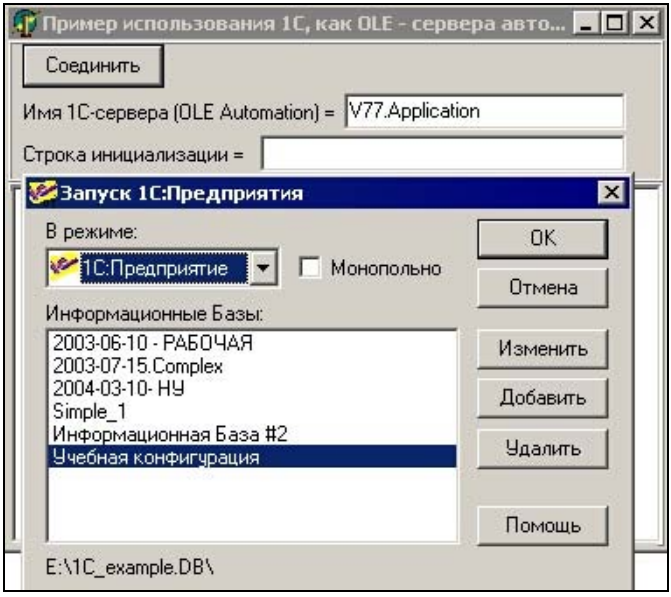



**Рис. 3.5.** Диалоговое окно выбора базы данных 1С

После загрузки и инициализации программы 1С ссылка на нее хранится в переменной `Ole1C`.

Следует помнить, что программа 1С:Предприятие (как и все создаваемые объекты OLE Automation), выступающая в качестве объекта OLE Automation в другой программе, будет находиться в памяти компьютера до удаления или изменения значения переменной, содержащей ее в качестве указателя на объект. Именно поэтому мы вынесли объявление переменной `Ole1C` "за пределы" обработчика `TForm1.Button1Click`.

Явных (видимых) признаков того, что мы запустили 1С, нет. Это связано с тем, что программа 1С:Предприятие в режиме сервера OLE Automation скрывает свое окно. В дальнейшем будут рассмотрены способы визуализации окна 1С.

Удостовериться в том, что программа 1С запущена, можно, запустив диспетчер задач (комбинация <Ctrl>+<Alt>+<Del>). В списке процессов (вкладка **Процессы**) можно увидеть строку **1cv7s.exe** (рис. 3.6) — это свидетельствует о том, что программа 1С активна.

После закрытия программы sp_Connect_to_1C_01.EXE автоматически выгрузится из памяти и программа 1С.

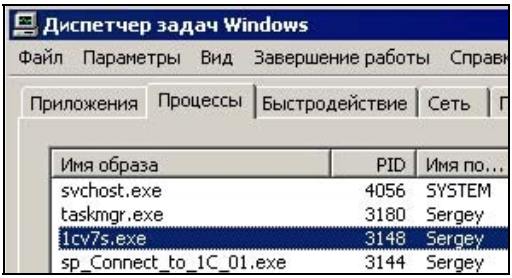



**Рис. 3.6.** Отображение программы 1С:Предприятие в списке запущенных процессов

# 3.2. Второй этап

Возьмем за основу предыдущий проект и дополним его.

***Примечание***

Исходные тексты находятся на сопроводительном компакт-диске книги в файле \Part_1\Chapter_03\Topic_3_2\sp_Connect_to_1C_02.PAS.

1. Поместим на панель `Panel1` еще одну кнопку `Button2` (`TButton`) и текстовое поле `Edit3` (`TEdit`).
2. Установим значения свойств:
   - `Button2.Caption = "Справочник"`
   - `Edit3.Text = "ФизЛица"`

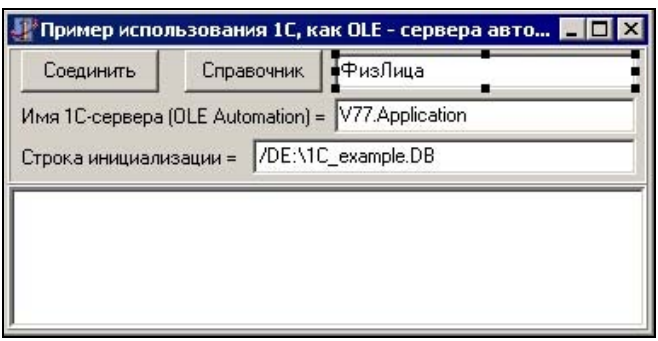



**Рис. 3.7.** Форма примера подключения к 1С (второй этап)

3. Сделаем двойной щелчок на кнопке `Button2` и изменим стандартный обработчик события (нажатия кнопки), как показано в листинге 3.3.

**Листинг 3.3. Считывание информации из справочника ФизЛица**

```
procedure TForm1.Button2Click(Sender: TObject);
Var
  OleFizLica: OleVariant;
begin
  OleFizLica:= Unassigned; //1
  ListBox1.Items.Clear;
  if not VarIsEmpty(Ole1C) then begin //2
     OleFizLica:=Ole1C.CreateObject('Справочник.'+Edit3.Text); //3
     if not VarIsEmpty(OleFizLica) then begin //4
       if OleFizLica.ВыбратьЭлементы>0 then begin //5
          while OleFizLica.ПолучитьЭлемент>0 do //6
           begin
            ListBox1.Items.Add(
                                Trim(OleFizLica.Код)+'  ' //7
                              + Trim(OleFizLica.Наименование)+' ' //8
                              + Trim(OleFizLica.ДатаРождения)+' ' //9
                              + Trim(OleFizLica.ДомашнийАдрес) //10
                               );
            Application.ProcessMessages;
          end;
       end;
     end;
  end;
  OleFizLica:= Unassigned; //11
end;
```

Переменная `OleFizLica` (типа `OleVariant`) служит указателем на объект 1С `Справочник.ФизЛица`.

Строки 1 и 11 (номера строк указаны в комментариях листинга):

```
OleFizLica:= Unassigned;
```

Здесь "обнуляется" значение переменной `OleFizLica`. Тем самым, если ранее был создан какой-либо OLE-объект, на который ссылалась эта переменная, он будет уничтожен. В данном случае это излишки кода обработчика; эти строки оставлены лишь в качестве иллюстрации.

Строка 2:

```
if not VarIsEmpty (Ole1C) then begin
```

Это проверка: значение переменной `Ole1C` не должно быть "пустым". То есть "непустое" значение переменной `Ole1C` означает (в контексте нашего проекта), что программа 1С запущена (как сервер OLE Automation) и связь с ней установлена.

Строка 3:

```
OleFizLica:=Ole1C.CreateObject('Справочник.'+Edit3.Text);
```

Здесь создается объект, который ссылается на спавочник с идентификатором `ФизЛица`. Если такой справочник существует, то в дальнейшем переменная `OleFizLica` будет содержать ссылку на него.

Строка 4:

```
if not VarIsEmpty (OleFizLica) then begin
```

Это проверка: значение переменной `OleFizLica` не должно быть "пустым". То есть "непустое" значение переменной `OleFizLica` означает (в контексте нашего проекта), что объект 1С `Справочник.ФизЛица` создан и можно обращаться к его свойствам и методам.

Строки 5 и 6:

```
if OleFizLica.ВыбратьЭлементы>0 then begin
while OleFizLica.ПолучитьЭлемент>0 do
```

Здесь вызываются методы объекта типа `Справочник`:

- `ВыбратьЭлементы` — осуществляет выборку элементов справочника;
- `ПолучитьЭлемент` — осуществляет позиционирование на каждом элементе справочника поочередно в цикле до тех пор, пока не вернет значение 0.

Строки 7—10 организуют обращение к свойствам и реквизитам текущего элемента справочника.

***Примечание***

Функция `Trim()` используется для того, чтобы конвертировать значение типа `Variant` (возвращаемое через механизм OLE) в строковое значение. Если не предусмотреть этого, то может последовать исключение (рис. 3.8 и 3.9).

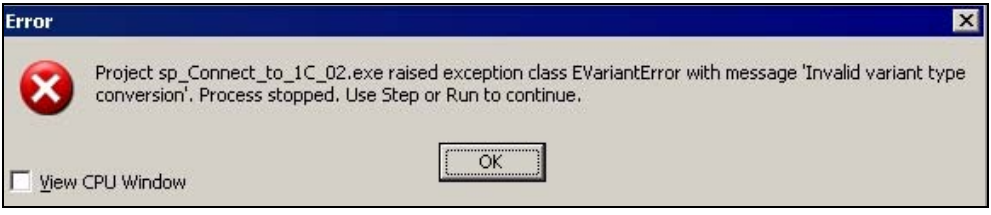



**Рис. 3.8.** Сообщение об исключении, вызванном некорректной операцией с переменной типа `Variant`

**Рис. 3.9.** Сообщение об исключении, вызванном некорректной операцией с переменной типа `Variant`

4. Сохраним проект в файле с именем sp_Connect_to_1C_02.DPR (или любым другим, на усмотрение читателя), откомпилируем его и запустим приложение sp_Connect_to_1C_02.EXE.
5. Нажмем кнопку **Соединить**, а после того как программа 1С загрузится (см. предыдущий пример), — кнопку **Справочник**.

Если все указано корректно (поле `Edit3.Text`), программа 1С создаст объект типа `Справочник.ФизЛица`, ссылка на который будет размещена в переменной `OleFizLica`. Затем информация из справочника будет считана и размещена в списке `ListBox1` (рис. 3.10).

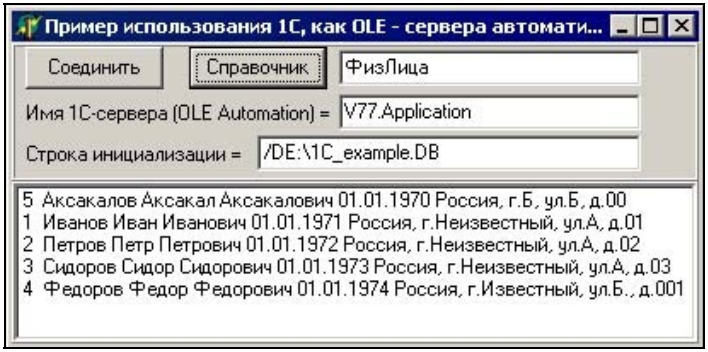



**Рис. 3.10.** Результат считывания информации из справочника "Список физических лиц"

### *Примечание*

Порядок считывания элементов справочника (при обращении к методу `ПолучитьЭлемент`) определяется порядком сортировки справочника "Список физических лиц", указанным в Конфигураторе (см. рис. 2.13).

В следующей главе более подробно рассмотрены вопросы считывания информации из базы данных 1С.

# Глава 4

# Доступ к информации 1С из внешних приложений

Прежде чем приступить к программированию, рассмотрим основные типы данных, применяемые в системе 1С для формирования информационных структур конфигураций (отражения свойств и методов объектов в предметной области).

Поддерживаются следующие типы данных:

- *базовые типы данных:*
  - `Числовой`
  - `Строковый`
  - `Дата`
- *агрегатные типы данных* — это специализированные типы данных системы 1С:Предприятие. К агрегатным типам данных, участвующим в формировании информационной структуры, относятся:
  - `Константа`
  - `Справочник`
  - `Перечисление`
  - `Документ`
  - `Регистр`
  - `ПланСчетов`
  - `Счет`
  - `ВидСубконто`
  - `Операция`
  - `БухгалтерскиеИтоги`
  - `ЖурналРасчетов`

- ВидРасчета
- ГруппаРасчетов
- Календарь
- Периодический

У агрегатных типов данных есть *атрибуты* (свойства агрегатного типа данных) и *методы* (действия, которые может выполнять агрегатный тип данных). Например, в *главе 3* мы обращались к методам ВыбратьЭлементы и ПолучитьЭлемент агрегатного типа данных Справочник

**ПРИМЕЧАНИЕ**

В системе 1С:Предприятие можно обращаться к именам типов данных, их атрибутам и методам как в русском, так и англоязычном написании (табл. 4.1).

***Таблица 4.1.*** *Агрегатные типы данных и их англоязычные синонимы в системе 1С:Предприятие*

| Агрегатный тип данных | Англоязычный синоним |
|---|---|
| Константа | Const |
| Справочник | Reference |
| Перечисление | Enum |
| Документ | Document |
| Регистр | Register |
| ПланСчетов | ChartOfAccounts |
| Счет | Account |
| ВидСубконто | SubcontoKind |
| Операция | Operation |
| БухгалтерскиеИтоги | BookkeepingTotals |
| ЖурналРасчетов | CalcJournal |
| ВидРасчета | CalculationKind |
| ГруппаРасчетов | CalculationGroup |
| Календарь | Calendar |
| Периодический | Periodic |

Типы данных в системе 1С:Предприятие имеют числовой эквивалент, или *код* (табл. 4.2).

***Таблица 4.2.*** *Коды типов данных в системе 1С:Предприятие*

| Тип данных | Код |
|---|---|
| Неопределенный | 0 |
| Числовой | 1 |
| Строковый | 2 |
| Дата | 3 |
| Перечисление | 10 |
| Справочник | 11 |
| Документ | 12 |
| Календарь | 13 |
| ВидРасчета | 14 |
| Счет | 15 |
| ВидСубконто | 16 |
| ПланСчетов | 17 |
| Внешний объект (прочие агрегатные объекты) | 100 |

***Примечание***

Более подробную характеристику и назначение всех указанных выше типов данных можно найти в любой специальной литературе по системе 1С:Предприятие (здесь не приводится).

В рамках данной книги мы рассмотрим способы обращения к информации различных типов данных 1С:Предприятие на конкретных примерах.

***Примечание***

Представленные ниже примеры исходных текстов — упрощенная иллюстрация к рассматриваемым вопросам. Назначение этих примеров — дать представление о возможном подходе при решении задач информационного обмена с системой 1С:Предприятие. При работе с реальными базами 1С может потребоваться их дальнейшая доработка.

# 4.1. Считывание информации из справочников

Используя созданную ранее конфигурацию 1С, рассмотрим вопросы доступа к информации из базы данных 1С с использованием механизма OLE Automation.

## 4.1.1. Реквизит справочника типа *Перечисление*

Опираясь на информацию, полученную в *главе 2*, запустим Конфигуратор 1С и откроем нашу конфигурацию (1С_example.DB).

Добавим в конфигурацию новое перечисление — "Принадлежность к полу" (с идентификатором `Пол`).

> ***Примечание***
>
> В терминологии 1С *перечисления* — это списки значений, задаваемые на этапе конфигурирования, которые не могут быть изменены при выполнении задачи.

Для создания нового перечисления нужно в конфигурации выделить с помощью мыши строку **Перечисления** и, вызвав всплывающее меню, выбрать в нем пункт **Новое Перечисление** (рис. 4.1).

На экране появится диалоговое окно настройки перечисления (рис. 4.2).

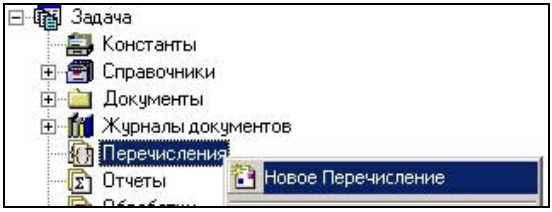



**Рис. 4.1.** Всплывающее меню Конфигуратора (новое перечисление)

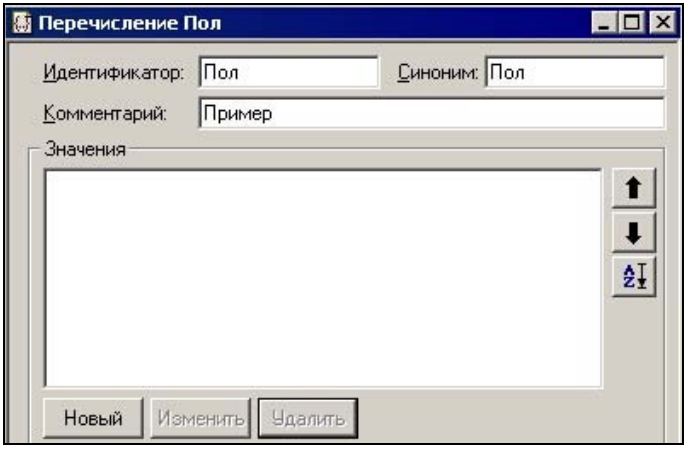



**Рис. 4.2.** Диалоговое окно настройки перечисления

***Примечание***

Идентификатор перечисления должен быть уникальным в контексте данной конфигурации. Все обращения через механизм OLE Automation к этому перечислению производятся с использованием этого уникального идентификатора.

Чтобы внести новое значение перечисления, нужно нажать кнопку **Новый** (см. рис. 4.2). На экране появится диалоговое окно настройки значения перечисления (рис. 4.3).

***Примечание***

Идентификатор каждого значения в контексте перечисления должен быть уникальным.

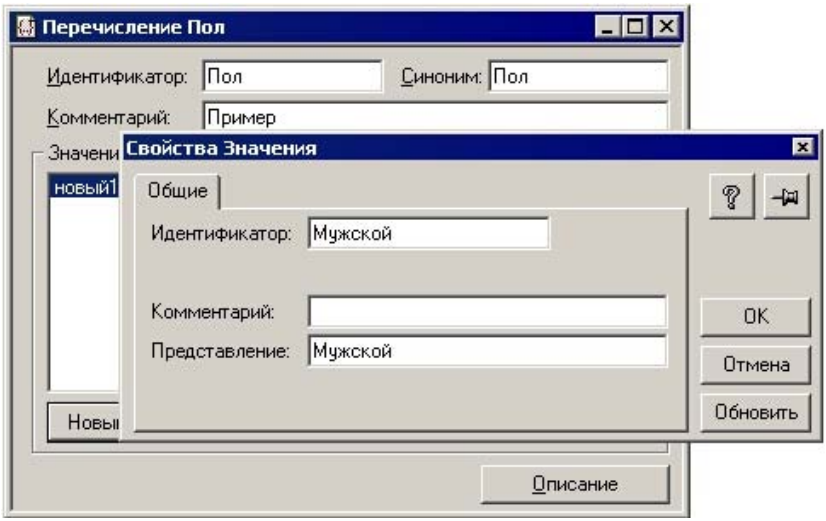



**Рис. 4.3.** Диалоговое окно ввода нового значения перечисления

В перечисление вида `Пол` внесем два значения:

- `Мужской`
- `Женский`

Точно так же создадим еще одно перечисление вида `СемейноеПоложение`. Заполним его значениями:

- `НикогдаНеСостоялВБраке`
- `СостоитВЗарегистрированномБраке`
- `СостоитВНеЗарегистрированномБраке`
- `ГражданскийБрак`

- Разведен
- Разошелся
- Вдовец

На рис. 4.4 представлено дерево конфигурации после внесенных изменений.

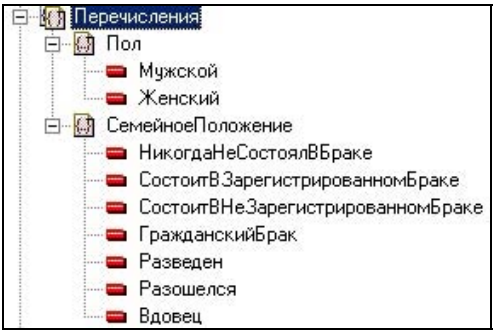



**Рис. 4.4.** Дерево конфигурации. Ветвь метаданных **Перечисления**

Внесем изменения в справочник `ФизЛица`. Для этого откроем ветвь конфигурации **Справочники** и сделаем двойной щелчок на строке **ФизЛица** (рис. 4.5).

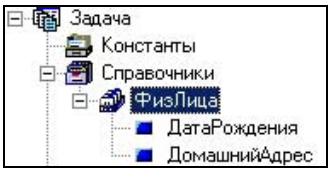



**Рис. 4.5.** Выбор строки для изменения справочника

На экране появится диалоговое окно настройки справочника `ФизЛица`. Внесем новый реквизит типа — `Перечисление.Пол` (рис. 4.6).

Точно так же внесем еще один реквизит — `СемейноеПоложение`. В результате список реквизитов справочника `ФизЛица` будет иметь вид, как на рис. 4.7.

Теперь нужно сохранить конфигурацию (см. рис. 2.17).

***Примечание***

Содержание (список файлов) папки 1C_example.DB осталось неизменным. Это связано с тем, что перечисления хранятся в файле 1Cv7.MD.

Созданные нами изменения в справочнике отражены в словаре данных (файл 1Cv7.DD, рис. 4.8).

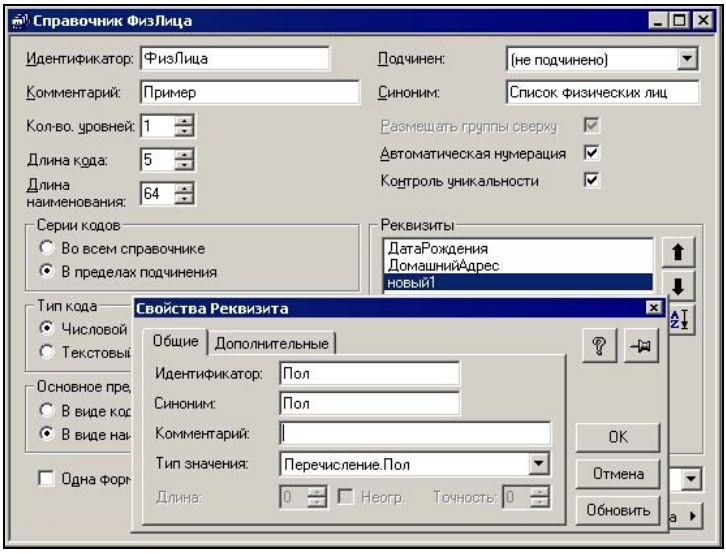



**Рис. 4.6.** Новый реквизит в справочнике ФизЛица — Пол

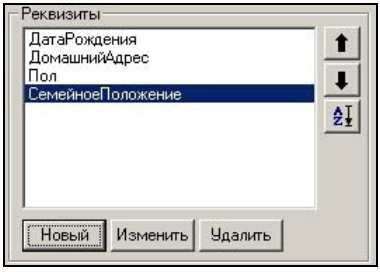


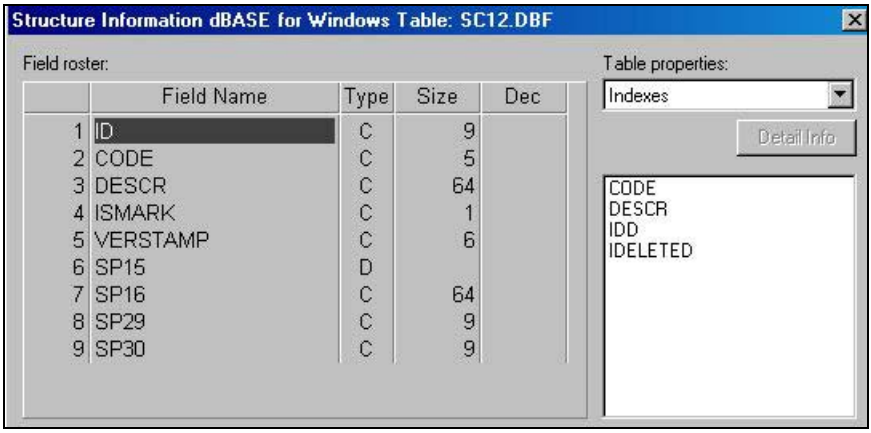

**Рис. 4.7.** Список реквизитов в справочнике ФизЛица

```
#==========================================================================
#==TABLE no 8      : Справочник ФизЛица
# Name     |Descr                          |Type[A/S/U]|DBTableName|ReUsable
T=SC12     |Справочник ФизЛица             |A          |SC12       |1
#-----Fields-------
# Name       |Descr                |Type|Length|Precision
F=ID         |ID object            |C   |9     |0
F=CODE       |object code          |C   |5     |0
F=DESCR      |object description   |C   |64    |0
F=ISMARK     |Flag Object is Marke|C   |1     |0
F=VERSTAMP   |Version stamp        |C   |6     |0
F=SP15       |(P)ДатаРождения      |D   |8     |0
F=SP16       |(P)ДомашнийАдрес     |C   |64    |0
F=SP29       |(P)Пол               |C   |9     |0
F=SP30       |(P)СемейноеПоложение|C   |9     |0
#----Indexes------
# Name      |Descr          |Unique|Indexed fields                                    |DBName
I=IDD       |of ID          |0     |ID                                                |IDD
I=CODE      |of CODE        |0     |CODE(UPPER)                                       |CODE
I=DESCR     |of DESCR       |0     |DESCR(UPPER)                                      |DESCR
```

**Рис. 4.8.** Содержание файла 1Cv7.DD. Структура справочника ФизЛица

Новые реквизиты (`Пол` и `СемейноеПоложение`) были размещены в справочнике `ФизЛица` (файл таблицы SC12.DBF). Имена полей в DBF-файле формируются системой 1С на основе внутреннего алгоритма. Как видите, здесь нет никаких указаний на то, каким образом связаны поля таблицы (`SP29` и `SP30`) со структурой данных, хранящейся в файле 1Cv7.MD, а также не отражена информация о том, какой именно тип данных имеют эти поля в контексте информационной структуры конфигурации 1С.

На рис. 4.9 представлена структура таблицы SC12.DBF, если открыть ее в Database Desktop (Delphi).




| | Field Name | Type | Size | Dec |
|---|---|---|---|---|
| 1 | ID | C | 9 | |
| 2 | CODE | C | 5 | |
| 3 | DESCR | C | 64 | |
| 4 | ISMARK | C | 1 | |
| 5 | VERSTAMP | C | 6 | |
| 6 | SP15 | D | | |
| 7 | SP16 | C | 64 | |
| 8 | SP29 | C | 9 | |
| 9 | SP30 | C | 9 | |

**Рис. 4.9.** Структура файл SC12.DBF в Database Desktop

Чтобы внести значения новых реквизитов, нужно запустить программу 1С:Предприятие (см. рис. 2.23). На рис. 4.10 отображено представление введенных данных в среде 1С.

| Код | Наименование | Дата... | Домашний адрес | Пол | Семейное положение |
|---|---|---|---|---|---|
| 5 | Аксакалов Аксакал Аксакалович | 01.01.70 | Россия, г.Б, ул.Б, д.00 | Мужской | Разведен (разведена) |
| 1 | Иванов Иван Иванович | 01.01.71 | Россия, г.Неизвестный, ул. | Мужской | Состоит в зарегистрированн |
| 2 | Петров Петр Петрович | 01.01.72 | Россия, г.Неизвестный, ул. | Мужской | Гражданский брак |
| 3 | Сидоров Сидор Сидорович | 01.01.73 | Россия, г.Неизвестный, ул. | Мужской | Вдовец (вдова) |
| 4 | Федоров Федор Федорович | 01.01.74 | Россия, г.Известный, ул.Б., | Мужской | Разошелся (разошлась) |
| 6 | Криогенова Фекла Маринадовна | 15.03.65 | Россия, г.N, ул., У, д. 005 | Женс | Никогда не состоял (не сост |
| | | | | Мужской | |
| | | | | Женский | |

**Рис. 4.10.** Справочник `ФизЛица`. Новые реквизиты типа `Перечисление`

На рис. 4.11 отображено представление введенных данных в DBF-файле (SC12.DBF), который соответствует справочнику `ФизЛица`. Как видите, по со-

держанию полей `SP29` и `SP30`, которые соответствуют данным типа `Перечисление`, невозможно сделать какие-либо выводы о реальной информации по каждому физическому лицу. Мы можем сделать предположения, лишь руководствуясь известной нам информацией о структуре, поскольку сами создали эту конфигурацию. В *разд. 4.4* мы рассмотрим вопрос идентификации значений перечислений вне контекста программы 1С:Предприятие.

Table : D:\1C_example.DB\SC12.DBF

| SC12 | ID | CODE | DESCR | ISMARK | VERSTAMP | SP15 | SP16 | SP29 | SP30 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | Иванов Иван Иванович | | 6 | 01.01.1971 | Россия, г.Неизвестный, ул | J | N |
| 2 | 2 | 2 | Петров Петр Петрович | | 4 | 01.01.1972 | Россия, г.Неизвестный, ул | J | P |
| 3 | 3 | 3 | Сидоров Сидор Сидорович | | 9 | 01.01.1973 | Россия, г.Неизвестный, ул | J | S |
| 4 | 4 | 4 | Федоров Федор Федорович | | 5 | 01.01.1974 | Россия, г.Известный, ул.Е | J | R |
| 5 | 5 | 5 | Аксакалов Аксакал Аксакал | | 5 | 01.01.1970 | Россия, г.Б, ул.Б, д.00 | J | Q |
| 6 | 6 | 6 | Криогенова Фекла Маринад | | 0 | 15.03.1965 | Россия, г.N, ул., У, д. 005 | K | M |

**Рис. 4.11.** Справочник `ФизЛица` — так отражена информация в файле SC12.DBF, если открыть его в Database Desktop

Рассмотрим пример, иллюстрирующий считывание и анализ информации с использованием данных типа `Перечисление` в среде Delphi.

Создадим новый проект с именем sp_LoadEnumFromRef.DPR (или любым другим, на усмотрение читателя). За основу проекта можно взять последний пример из предыдущей главы.

> ***Примечание***
>
> Исходный текст проекта представлен на сопроводительном компакт-диске книги в файле \Part_1\Chapter_04\Topic_4_1\Item_4_1_1\sp_LoadEnumFromRef.PAS.

На рис. 4.12 приведена Delphi-форма примера.

Способ соединения с 1С в качестве сервера OLE Automation был рассмотрен в предыдущей главе и более на этом не будем акцентрировать внимание.

Прежде чем приступить к написанию кода, примем к сведению, что в качестве имени значения перечисления обязательно должно выступать полное имя конкретного вида и значения перечисления, как оно объявлено в Конфигураторе. Вид и значение перечисления записываются через точку после ключевого слова `Перечисление`, т. е. полное имя значения перечисления записывается следующим образом:

`Перечисление.<ID_Переч>.<ID_ЗначенияПереч>`

где `<ID_Переч>` — идентификатор перечисления, `<ID_ЗначенияПереч>` — идентификатор конкретного значения данного перечисления.

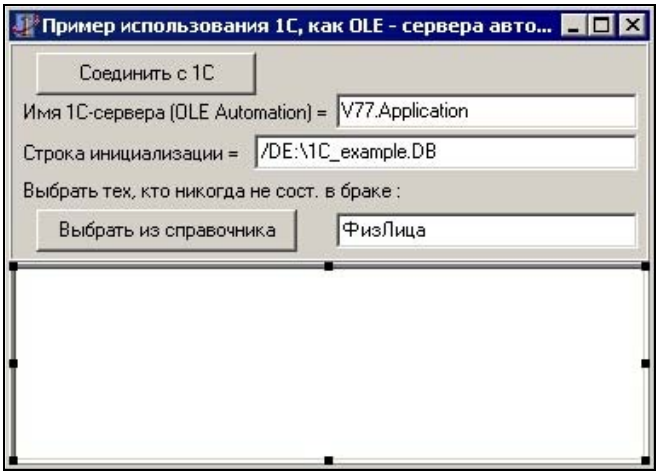



**Рис. 4.12.** Форма примера, иллюстрирующего работу с данными типа `Перечисление`

Объект `Перечисление` является частью глобального контекста системы 1С:Предприятие. Это значит, что мы можем обратиться к этому типу метаданных через ссылку на программу 1С как сервер OLE Automation (переменная `Ole1C`).

***Примечание***

Англоязычный синоним ключевого слова `Перечисление` — `Enum`.

Поставим перед собой задачу прочитать информацию из справочника `ФизЛица` только о тех физических лицах, кто никогда не состоял в браке. Вспомним, что для этих целей в Конфигураторе было создано перечисление `СемейноеПоложение`, одно из значений которого — `НикогдаНеСостоялВБраке` (см. рис. 4.4). Вспомним также, что в справочнике `ФизЛица` для отражения этой информации был введен специальный реквизит `СемейноеПоложение` (не путать с одноименным перечислением как объектом метаданных), см. рис. 4.7.

В контексте встроенного языка 1С можно сравнивать значения одного типа непосредственно, например, как показано в листинге 4.1.

**Листинг 4.1. Сравнение объектов в 1С**

```
Если
Спр.СемейноеПоложение=Перечисление.СемейноеПоложение.НикогдаНеСостоялВБра
ке Тогда
...
КонецЕсли;
```

При доступе через OLE Automation такой подход будет некорректным. Нужно сравнивать уникальные идентификаторы, которые представляют собой строки или целые числа.

Вооружившись этими сведениями, изменим обработчик события (см. листинг 3.3) "нажатие кнопки **Выбрать из справочника** (`Button2`)", как показано в листинге 4.2.

**Листинг 4.2. Выбор из справочника информации о лицах, никогда не состоявших в браке**

```
procedure TForm1.Button2Click(Sender: TObject);
Var
  OleBuf, // Вспомогательная переменная
  OleFizLica: OleVariant; // Ссылка на справочник "ФизЛица"
// Идентификатор реквизита значения справочника "СемейноеПоложение":
  idEnumX,
// Идентификатор значения перечисления "СемейноеПоложение" (метаданные):
  idEnumMeta: string;
begin
  ListBox1.Items.Clear;
  if not VarIsEmpty(Ole1C) then begin
     // Получаем ссылку на перечисление СемейноеПоложение
     OleBuf:= Ole1C.Перечисление.СемейноеПоложение;
     // Читаем идентификатор соответствующего значения
     // соответствующего перечисления
     idEnumMeta:=Trim(OleBuf.НикогдаНеСостоялВБраке.Идентификатор);
     // Создаем справочник "ФизЛица" и получаем на него ссылку
     OleFizLica:=Ole1C.CreateObject('Справочник.'+Edit3.Text);
     if not VarIsEmpty(OleFizLica) then begin
        if OleFizLica.ВыбратьЭлементы>0 then begin
           while OleFizLica.ПолучитьЭлемент>0 do
            begin
             // Читаем идентификатор значения реквизита
             // справочника "СемейноеПоложение"
             idEnumX:=Trim(OleFizLica.СемейноеПоложение.Идентификатор);
             // Сравниваем идентификаторы
             if idEnumX = idEnumMeta then begin
               ListBox1.Items.Add(
                              Trim(OleFizLica.Код)+' | '
```

```
                                + Trim(OleFizLica.Наименование)+' | '
                                + Trim(OleFizLica.ДатаРождения)+' | '
                            + Trim(OleFizLica.Пол.Идентификатор)+' | '
                                 + Trim(OleFizLica.ДомашнийАдрес)
                                      );
            end;
            Application.ProcessMessages;
          end;
        end;
      end;
  end;
end;
```

Поясним текст обработчика.

В строке

```
OleBuf:= Ole1C.Перечисление.СемейноеПоложение;
```

получаем ссылку на перечисление `СемейноеПоложение` как объект.

В строке

```
idEnumMeta:=Trim(OleBuf.НикогдаНеСостоялВБраке.Идентификатор);
```

считывается идентификатор соответствующего *значения перечисления* (как объекта) из глобального контекста конфигурации.

В строке

```
idEnumX:=Trim(OleFizLica.СемейноеПоложение.Идентификатор);
```

считывается идентификатор *значения реквизита справочника* (как объекта).

В строке

```
if idEnumX = idEnumMeta then begin
```

сравнивается идентификатор значения реквизита `СемейноеПоложение` справочника `ФизЛица` с идентификатором значения `НикогдаНеСостоялВБраке` перечисления `СемейноеПоложение`.

В строке

```
+ Trim(OleFizLica.Пол.Идентификатор)+' | '
```

выполняется обращение к идентификатору значения реквизита `Пол` (как объекта) справочника `ФизЛица`.

На рис. 4.13 показан результат считывания информации из справочника `ФизЛица`.

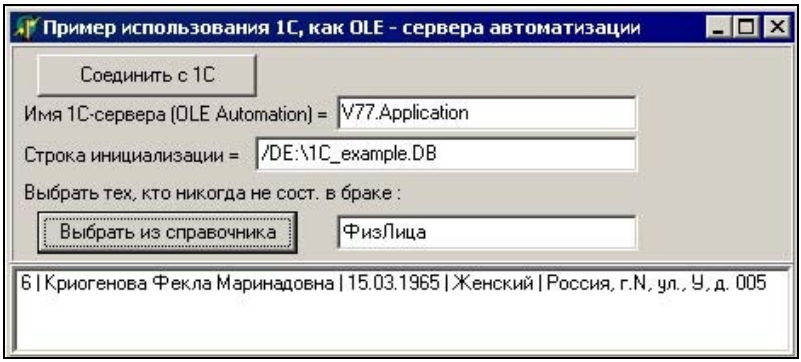



**Рис. 4.13.** Результат фильтрации по реквизиту `СемейноеПоложение`

***Примечание***

В данном примере мы рассмотрели способ считывания непериодических реквизитов справочника.

Вот так выглядит обработчик события "нажатие кнопки" на встроенном языке 1С в контексте модуля `Обработка` (листинг 4.3).

**Листинг 4.3. Выбор из справочника `ФизЛица` лиц, никогда не состоявших в браке (язык программирования 1С)**

```
Процедура Выполнить()
  СпрФизЛица = СоздатьОбъект("Справочник.ФизЛица");
  СпрФизЛица.ВыбратьЭлементы();
  Пока СпрФизЛица.ПолучитьЭлемент()>0 Цикл
    ЗначРеквизСправочн = СпрФизЛица.СемейноеПоложение;
    ЗначПеречисления    = Перечисле-
ние.СемейноеПоложение.НикогдаНеСостоялВБраке;
    Если ЗначРеквизСправочн = ЗначПеречисления Тогда
      Сообщить(СпрФизЛица.Код + " | "
              + СпрФизЛица.Наименование + " | "
              + СпрФизЛица.ДатаРождения + " | "
              + СпрФизЛица.Пол + " | "
              + СпрФизЛица.ДомашнийАдрес);
    КонецЕсли;
  КонецЦикла;
КонецПроцедуры
```

А это результат выполнения:

```
6 | Криогенова Фекла Маринадовна | 15.03.65 | Женский | Россия, г.N, ул., У, д. 005
```

## 4.1.2. Реквизит справочника типа *Строка неограниченной длины*

Для того чтобы рассмотреть способ доступа к реквизитам типа `Строка неограниченной длины` (аналог поля `МЕМО`), доработаем нашу конфигурацию.

Опираясь на информацию, полученную в *главе 2*, запустим Конфигуратор 1С и откроем нашу конфигурацию (1C_example.DB).

Откроем окно настройки справочника (см. рис. 4.5 и 4.6) и внесем в справочник новый реквизит, как на рис. 4.14.

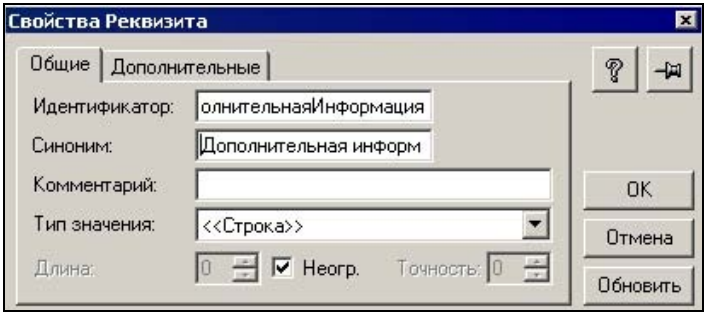



**Рис. 4.14.** Реквизит типа `Строка неограниченной длины`

После этой операции список реквизитов справочника `ФизЛица` будет таким, как показано на рис. 4.15.

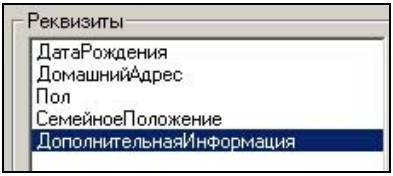



**Рис. 4.15.** Список реквизитов справочника `ФизЛица`

Поскольку значения реквизита данного типа (строка неограниченной длины) могут содержать специальные символы форматирования текста (перевод строки и возврат каретки), эту информацию невозможно будет корректно вносить в контексте списка (см. рис. 4.10). Чтобы решить эту проблему, нужно создать форму редактирования элементов справочника `ФизЛица`. Для этого в диалоговом окне создания и настройки справочника (см. рис. 2.13) нужно нажать кнопку **Форма элемента**. На экране появится диалоговое окно создания формы редактирования элемента справочника (рис. 4.16).

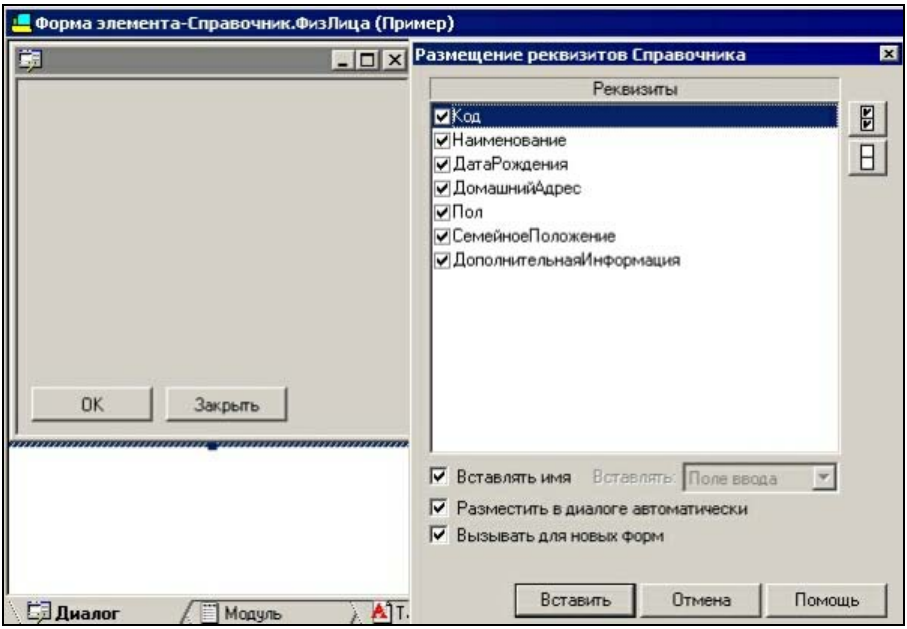



**Рис. 4.16.** Разработка формы для редактирования элементов справочника `ФизЛица`

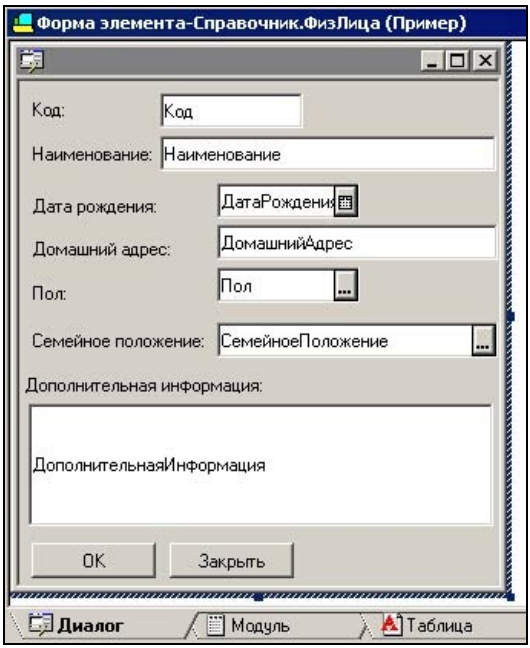



**Рис. 4.17.** Форма редактирования элементов справочника `ФизЛица`

Изучение методов разработки форм в системе 1С:Предприятие не входит в задачу этой книги, поэтому просто нажмем кнопку **Вставить** в этом окне. После того как соответствующие элементы управления автоматически разместятся в форме, можно расположить их по своему усмотрению (с помощью мыши и клавиатуры). Например так, как показано на рис. 4.17.

Затем нужно щелчком левой кнопки мыши выделить элемент управления **ДополнительнаяИнформация**, щелчком правой кнопки мыши вызвать всплывающее меню, выбрать в нем пункт **Свойства**, в открывшемся диалоговом окне настройки свойств элемента управления перейти на вкладку **Тип** (рис. 4.18) и установить флажок **Многострочный**.

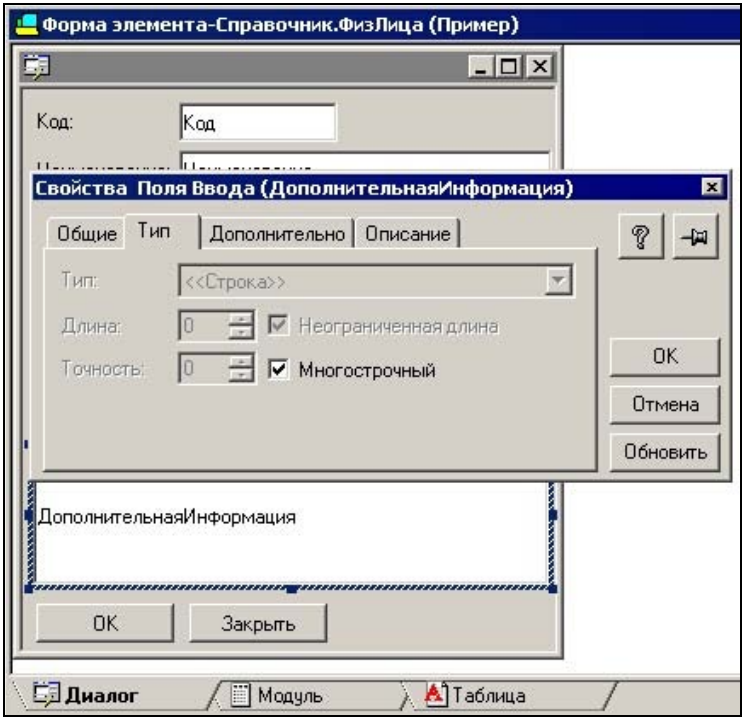



**Рис. 4.18.** Окно редактирования свойств элемента управления **ДополнительнаяИнформация**

Затем нужно закрыть форму редактирования элемента справочника и выбрать способ редактирования элементов справочника **В диалоге** (рис. 4.19).

Теперь нужно сохранить сделанные в конфигурации изменения (см. рис. 2.17).

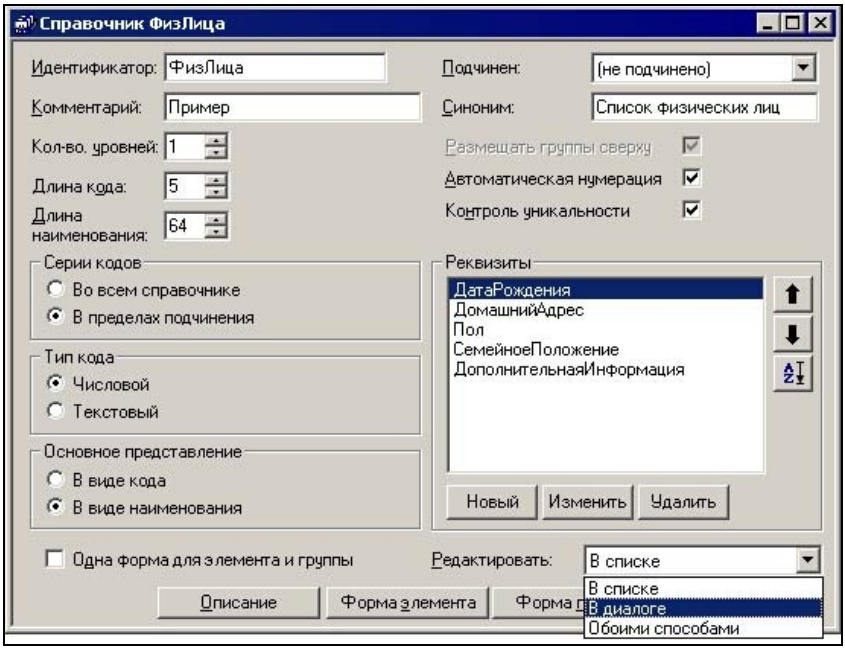



**Рис. 4.19.** Выбор способа редактирования элемента справочника

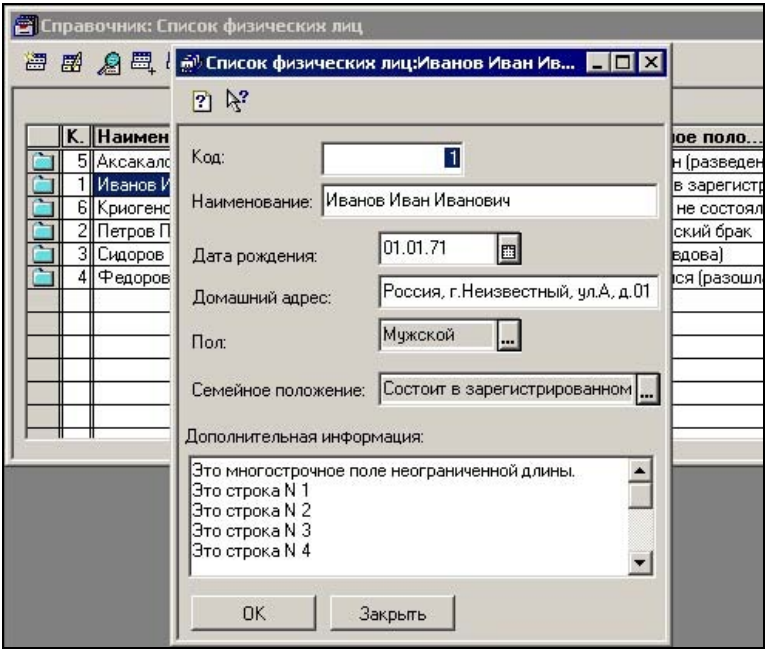



**Рис. 4.20.** Редактирование элемента справочника в диалоге

***Примечание***

Если открыть словарь данных (файл 1Cv7.DD), то можно увидеть, что содержание файла не изменилось. То есть несмотря на то, что в справочник `ФизЛица` был внесен новый реквизит, это не было отражено в словаре данных.

Запустим программу 1С:Предприятие, откроем справочник `ФизЛица` и сделаем двойной щелчок на строке с кодом 1. На экране появится созданная нами форма редактирования (рис. 4.20).

Внесем в поле **Дополнительная информация** несколько строк текста (см. рис. 4.20) и нажмем кнопку **OK**. Эта информация отобразится в списке так, как показано на рис. 4.21.

Справочник: Список физических лиц

| К. | Наименование | Дата... | Домашн... | Пол | Семейное... | Дополнительная информация |
|---|---|---|---|---|---|---|
| 5 | Аксакалов Аксакал Ак | 01.01.70 | Россия, г.Б, у | Мужской | Разведен (раз | |
| 1 | Иванов Иван Иванович | 01.01.71 | Россия, г.Не | Мужской | Состоит в зар | Это многострочное поле неограниченной длины. |
| 6 | Криогенова Фекла Ма | 15.03.65 | Россия, г.N, у | Женский | Никогда не сс | |
| 2 | Петров Петр Петрович | 01.01.72 | Россия, г.Не | Мужской | Гражданский | |
| 3 | Сидоров Сидор Сидоро | 01.01.73 | Россия, г.Не | Мужской | Вдовец (вдова | |
| 4 | Федоров Федор Федо | 01.01.74 | Россия, г.Из | Мужской | Разошелся (ра | |

**Рис. 4.21.** Так выглядит в списке реквизит типа `Строка неограниченной длины`

Для того чтобы понять, как именно система 1С:Предприятие размещает информацию для реквизитов типа `Строка неограниченной длины`, с помощью любого просмотрщика DBF-файлов откроем файл SC12.DBF и убедимся, что его содержание не изменилось (рис. 4.22).

Теперь откроем файл 1SBLOB.DBF (рис. 4.23).

Содержание таблицы :
SC12.DBF

Открыть таблицу

| ID | CODE | DESCR | ISMA | VERS | SP15 | SP16 | SP29 | SP30 |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | Иванов Иван Иванович | | H | 01.01.1971 | Россия, г.Неизвестный, ул. | J | N |
| 2 | 2 | Петров Петр Петрович | | 4 | 01.01.1972 | Россия, г.Неизвестный, ул. | J | P |
| 3 | 3 | Сидоров Сидор Сидорович | | 9 | 01.01.1973 | Россия, г.Неизвестный, ул. | J | S |
| 4 | 4 | Федоров Федор Федорович | | 5 | 01.01.1974 | Россия, г.Известный, ул.Б., | J | R |
| 5 | 5 | Аксакалов Аксакал Аксакалович | | 6 | 01.01.1970 | Россия, г.Б, ул.Б, д.00 | J | Q |
| 6 | 6 | Криогенова Фекла Маринадовна | | 4 | 15.03.1965 | Россия, г.N, ул., У, д. 005 | K | M |

**Рис. 4.22.** Содержание файла SC12.DBF

Содержание таблицы :
1SBLOB.DBF

Открыть таблицу

| FIELDID | OBJID | BLOCKNO | BLOCK |
|---|---|---|---|
| V | 1 | 0 | ef Это многострочное поле неограниченной длины.\|Это строка N 1\|Это строк |
| V | 1 | 1 | а N 2\|Это строка N 3\|Это строка N 4\|Это строка N 5\|Это строка N 6\|Это строк |
| V | 1 | 2 | а N 7\|Это строка N 8\|Это строка N 9\|Это строка N 10\|Это строка N 11\|Это стр |
| V | 1 | 3 | ока N 12\|Это строка N 1то строка N 9\|Это строка N 10\|Это строка N 11\|Это стр |

**Рис. 4.23.** Содержание файла 1SBLOB.DBF

### *Примечание*

Система 1С:Предприятие хранит строки неограниченной длины в файле 1SBLOB.DBF (табл. 4.3) блоками по 80 символов в каждом.

***Таблица 4.3.*** *Структура файла 1SBLOB.DBF*

| Field Name | Type | Size | Dec |
|---|---|---|---|
| FIELDID | C | 4 | |
| OBJID | C | 9 | |
| BLOCKNO | N | 4 | 0 |
| BLOCK | C | 80 | |

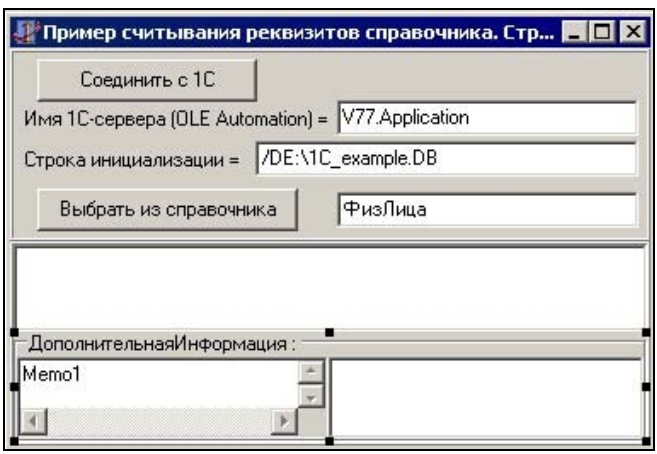



**Рис. 4.24.** Форма примера, иллюстрирующего работу со строками неограниченной длины

Посмотрим, что вернет нам программа 1С:Предприятие (как сервер OLE Automation), когда мы обратимся к реквизиту `ДополнительнаяИнформация` из внешнего приложения. Для этого создадим новый проект в среде Delphi (можно использовать последний пример). Поместим в форму еще два компонента: `Memo1` (`TMemo`) и `ListBox2 (TListBox)` (рис. 4.24).

Изменим обработчик события "нажатие кнопки **Выбрать из справочника**" следующим образом (листинг 4.4).

**Листинг 4.4. Считывание реквизита типа `Строка неограниченной длины` из справочника**

```
procedure TForm1.Button2Click(Sender: TObject);
Var
  OleFizLica: OleVariant;
begin
  ListBox1.Items.Clear;
  ListBox2.Items.Clear;
  Memo1.Lines.Clear;
  if not VarIsEmpty(Ole1C) then begin
     OleFizLica:=Ole1C.CreateObject('Справочник.'+Edit3.Text);
     if not VarIsEmpty(OleFizLica) then begin
        if OleFizLica.ВыбратьЭлементы>0 then begin
           while OleFizLica.ПолучитьЭлемент>0 do
            begin
             // Отберем только одну строку (с кодом 1),
             // где мы ввели значение
             // реквизита "ДополнительнаяИнформация"
             if Trim(OleFizLica.Код) = '1' then begin
                ListBox1.Items.Add(
                                   Trim(OleFizLica.Код)+' | '
                                 + Trim(OleFizLica.Наименование)+' | '
                          + Trim(OleFizLica.ДополнительнаяИнформация)
                                  );
                Memo1.Text:=Trim(OleFizLica.ДополнительнаяИнформация);
          ListBox2.Items.Text:=Trim(OleFizLica.ДополнительнаяИнформация);
             end;
             Application.ProcessMessages;
           end;
```

```
        end;
      end;
  end;
end;
```

Здесь никаких пояснений к тексту обработчика не требуется.

На рис. 4.25 представлен результат считывания. Сравните его с рис. 4.20 и 4.21.

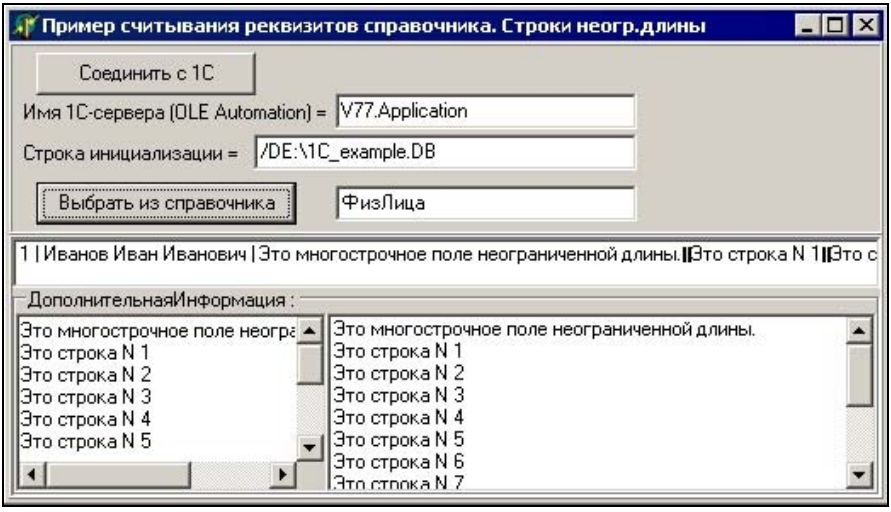



**Рис. 4.25.** Результат считывания реквизита `ДополнительнаяИнформация`

***Примечание***

Исходный текст этого примера есть на сопроводительном компакт-диске книги в файле Part_1\Chapter_04\Topic_4_1\Item_4_1_2\sp_LoadUnlimitStringFromRef.PAS.

В обработчике (листинг 4.4) для поиска одного элемента справочника по коду был использован прямой перебор всего справочника, что не может быть эффективным в общем случае. У агрегатного типа данных `Справочник` есть метод поиска элемента справочника по коду — `НайтиПоКоду` (или `FindByCode`). Его синтаксис:

`НайтиПоКоду(`*`Код, ФлагПоиска`*`)`

Параметры: *`Код`* — выражение со значением искомого кода; *`ФлагПоиска`* — 0 (поиск во всем справочнике вне зависимости от родителя), 1 (поиск внутри установленного подчинения, т. е. родителя), 2 (поиск по полному коду через

разделитель). Метод возвращает 1, если действие выполнено, и 0, если действие не выполнено (элемент не найден).

Перепишем обработчик (листинг 4.4) с учетом использования метода справочника `НайтиПоКоду` (листинг 4.5).

**Листинг 4.5. Использование метода справочника `НайтиПоКоду`**

```
procedure TForm1.Button2Click(Sender: TObject);
Var
  OleFizLica : OleVariant;
begin
  ListBox1.Items.Clear;
  ListBox2.Items.Clear;
  Memo1.Lines.Clear;
  if not VarIsEmpty(Ole1C) then begin
     OleFizLica:=Ole1C.CreateObject('Справочник.'+Edit3.Text);
     if not VarIsEmpty(OleFizLica) then begin
        // Вызываем метод справочника
        if OleFizLica.НайтиПоКоду(1,0)>0 then begin
           ListBox1.Items.Add(
                              Trim(OleFizLica.Код)+' | '
                            + Trim(OleFizLica.Наименование)+' | '
                            + Trim(OleFizLica.ДополнительнаяИнформация)
                             );
           Memo1.Text:=Trim(OleFizLica.ДополнительнаяИнформация);
       ListBox2.Items.Text:=Trim(OleFizLica.ДополнительнаяИнформация);
        end;
     end;
  end;
end;
```

После компиляции и выполнения примера результат будет таким же, как и на рис. 4.25.

***Примечание***

Исходные тексты этого примера есть на сопроводительном компакт-диске в файле \Part_1\Chapter_04\Topic_4_1\Item_4_1_2\sp_LoadUnlimitStringFromRef_01.PAS.

## 4.1.3. Реквизит справочника типа *Справочник*

Вновь доработаем нашу конфигурацию. Внесем еще два новых справочника без реквизитов (значения всех свойств — по умолчанию, длина наименования — 64 символа):

- "Страны мира" (идентификатор — `СтраныМира`);
- "Города стран мира" (идентификатор — `ГородаСтранМира`).

Справочник `ГородаСтранМира` будет подчинен справочнику `СтраныМира`, а справочник `СтраныМира` будет владельцем справочника `ГородаСтранМира`. Эти отношения между справочниками устанавливаются в диалоговом окне настройки подчиненного справочника (рис. 4.26).

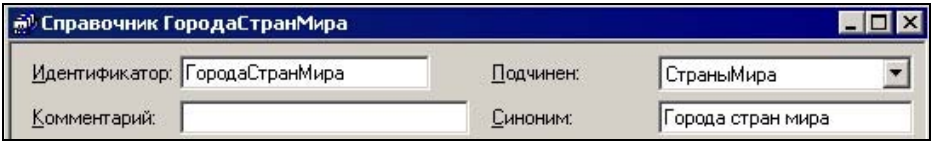



**Рис. 4.26.** Справочник `ГородаСтранМира` является подчиненным справочнику `СтраныМира`

В терминах Delphi справочник `СтраныМира` называется *главной таблицей* (master table), а справочник `ГородаСтранМира` — *подчиненной таблицей* (detail table).

На рис. 4.27 представлено дерево конфигурации после внесенных изменений.

**Рис. 4.27.** Ветвь справочников конфигурации после создания двух новых справочников

Сохраним изменения в конфигурации, запустим программу 1С:Предприятие и откроем справочник-владелец "Страны мира" (рис. 4.28).

После заполнения справочника `СтраныМира` нужно выбрать строку справочника-владельца и, вызвав всплывающее меню, выбрать пункт **Подчиненный справочник** (рис. 4.29).

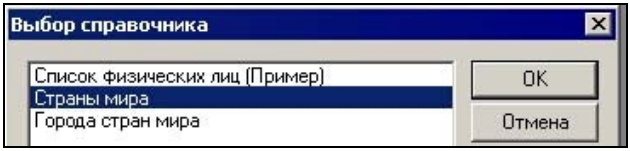



**Рис. 4.28.** Выбираем справочник-владелец

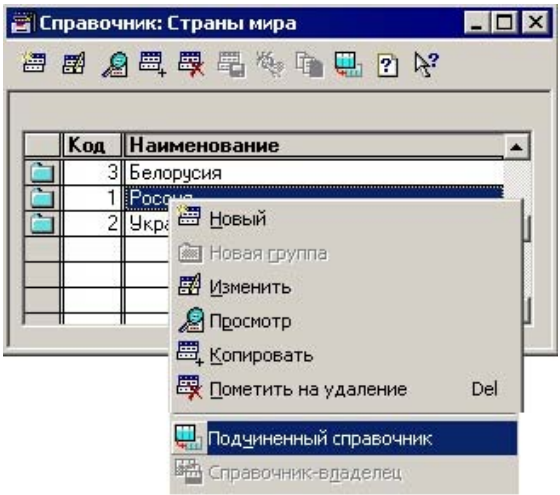



**Рис. 4.29.** Чтобы работать с подчиненным справочником, нужно открыть его из всплывающего меню владельца

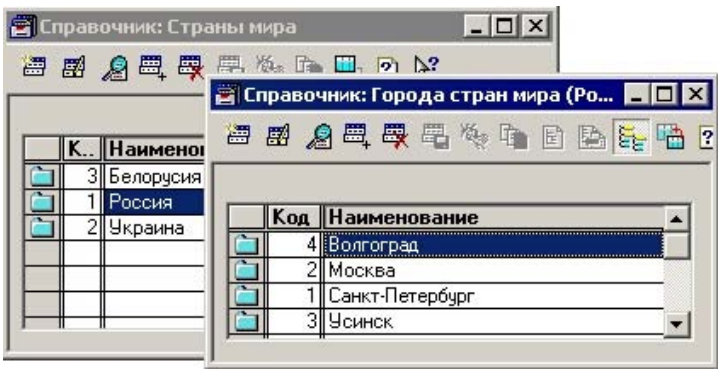



**Рис. 4.30.** Открыт подчиненный справочник

Откроется список подчиненного справочника.

Поочередно выбирая разные страны мира, заполним списки городов (рис. 4.30).

На рис. 4.31 и 4.32 представлена структура обеих таблиц так, как она отражена в файле 1Cv7.DD.

```
#==============================================================================
#==TABLE no 9      : Справочник СтраныМира
# Name    |Descr                          |Type[A/S/U]|DBTableName|ReUsable
T=SC33    |Справочник СтраныМира          |A          |SC33       |1
#-----Fields-------
# Name      |Descr                |Type|Length|Precision
F=ID        |ID object            |C   |9     |0
F=CODE      |object code          |C   |5     |0
F=DESCR     |object description   |C   |64    |0
F=ISMARK    |Flag Object is Marke|C   |1     |0
F=VERSTAMP  |Version stamp        |C   |6     |0
#----Indexes------
# Name     |Descr          |Unique|Indexed fields                                    |DBName
I=IDD      |of ID          |0     |ID                                                |IDD
I=CODE     |of CODE        |0     |CODE(UPPER)                                       |CODE
I=DESCR    |of DESCR       |0     |DESCR(UPPER)                                      |DESCR
#
```

**Рис. 4.31.** Содержание файла 1Cv7.DD. Структура справочника-владельца `СтраныМира`

```
#==============================================================================
#==TABLE no 10     : Справочник ГородаСтранМира
# Name    |Descr                          |Type[A/S/U]|DBTableName|ReUsable
T=SC35    |Справочник ГородаСтранМира     |A          |SC35       |1
#-----Fields-------
# Name      |Descr                |Type|Length|Precision
F=ID        |ID object            |C   |9     |0
F=CODE      |object code          |C   |5     |0
F=DESCR     |object description   |C   |64    |0
F=PARENTEXT |Parent in other tabl|C   |9     |0
F=ISMARK    |Flag Object is Marke|C   |1     |0
F=VERSTAMP  |Version stamp        |C   |6     |0
#----Indexes------
# Name     |Descr          |Unique|Indexed fields                                    |DBName
I=IDD      |of ID          |0     |ID                                                |IDD
I=PCODE    |of PARENT and  |0     |PARENTEXT,CODE(UPPER)                             |PCODE
I=PDESCR   |of PARENT and  |0     |PARENTEXT,DESCR(UPPER)                            |PDESCR
I=CODE     |of CODE        |0     |CODE(UPPER)                                       |CODE
I=DESCR    |of DESCR       |0     |DESCR(UPPER)                                      |DESCR
```

**Рис. 4.32.** Содержание файла 1Cv7.DD.
Структура справочника-владельца `ГородаСтранМира`

Как видите, в подчиненном справочнике `ГородаСтранМира` (файл SC35.DBF) для связи со справочником-владельцем `СтраныМира` выделено отдельное поле `PARENTEXT`.

Если открыть файл SC35.DBF (`ГородаСтранМира`), то (руководствуясь известной нам информацией о конфигурации) мы увидим, что поле `PARENTEXT` содержит коды соответствующих строк из таблицы SC33.DBF (`СтраныМира`).

Попробуем прочитать информацию из этих двух справочников, используя механизм OLE Automation. Для этого создадим новый проект (рис. 4.34).

Содержание таблицы :

SC35

Открыть таблицу

| ID | CODE | DESCR | PARENTEXT |
|---|---|---|---|
| 1 | 1 | Санкт-Петербург | 1 |
| 2 | 2 | Москва | 1 |
| 3 | 3 | Усинск | 1 |
| 4 | 4 | Волгоград | 1 |
| 5 | 1 | Одесса | 2 |
| 6 | 2 | Киев | 2 |
| 7 | 3 | Ильичевск | 2 |
| 8 | 4 | Донецк | 2 |
| 9 | 1 | Минск | 3 |
| A | 2 | Гомель | 3 |
| B | 3 | Брест | 3 |
| C | 4 | Барановичи | 3 |

**Рис. 4.33.** Содержание файла SC35.DBF, соответствующего справочнику `ГородаСтранМира`

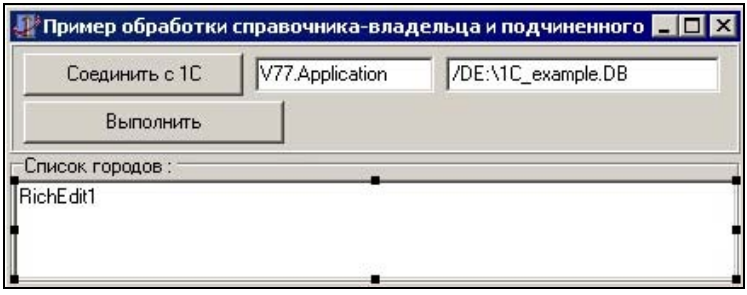



**Рис. 4.34.** Форма примера обработки справочника-владельца и подчиненного справочника

Рассмотрим обработчик события "нажатие кнопки **Выполнить**" (листинг 4.6).

**Листинг 4.6. Считывание информации из справочников "владелец" и "подчиненный"**

```
procedure TForm1.Button2Click(Sender: TObject);
Var
 iStr,iGor : integer;
 idOwner : string;// Код владельца для элемента "справ.ГородаСтранМира"
 OleStr,          // Ссылка на справочник "СтраныМира"
 OleGor : OleVariant;// Ссылка на справочник "ГородаСтранМира"
```

```
begin
  RichEdit1.Lines.Clear;
  if not VarIsEmpty(Ole1C) then begin
     OleStr:=Ole1C.CreateObject('Справочник.СтраныМира');
     if not VarIsEmpty(OleStr) then begin
        OleGor:=Ole1C.CreateObject('Справочник.ГородаСтранМира');
        if not VarIsEmpty(OleGor) then begin
           //Устанавливаем порядок сортировки справочника СтраныМира по кодам
           OleStr.ПорядокКодов;
           if OleStr.ВыбратьЭлементы>0 then begin
              while OleStr.ПолучитьЭлемент>0 do
               begin
                //Выводим код и наименование страны
                RichEdit1.Lines.Add(
                                    Trim(OleStr.Код)+'.'
                                  + Trim(OleStr.Наименование)
                                     );
// Устанавливаем порядок сортировки справочника "ГородаСтранМира"
// по наименованиям
                OleGor.ПорядокНаименований;
                if OleGor.ВыбратьЭлементы(0)>0 then begin
                   while OleGor.ПолучитьЭлемент>0 do
                    begin
// С помощью свойства "Владелец" подчиненного справочника
// получаем код строки-владельца для данного элемента справочника
                     //ГородаСтранМира
                     idOwner:=Trim(OleGor.Владелец.Код);
// Сравниваем полученный код с кодом текущего элемента
// справочника "СтраныМира"
                     if idOwner=Trim(OleStr.Код) then begin
//Выводим код и наименование города соотв.страны
                        RichEdit1.Lines.Add('   '
                                       + Trim(OleGor.Код)+'.'
                                       + Trim(OleGor.Наименование)
                                            );
                     end;
                     Application.ProcessMessages;
```

```
                end;
              end;
              Application.ProcessMessages;
            end;
          end;
        end;
      end;
    end;
  end;
```

Здесь мы организовали тривиальное считывание информации по циклу из справочника-владельца `СтраныМира`, а внутри этого цикла — внутренний цикл, где считывается и фильтруется информация из подчиненного справочника `ГородаСтранМира`.

Единственное, что заслуживает внимания — это использование таких методов объекта агрегатного типа `Справочник`, как `ПорядокКодов` и `ПорядокНаименований`, и свойства `Владелец`. Оба метода предназначены для того, чтобы изменить порядок сортировки элементов справочника при выборке. По умолчанию выборка элементов справочника производится в порядке основного представления справочника, задаваемого в Конфигураторе.

Если посмотреть на структуру таблиц, отраженную в файле 1Cv7.DD, то видно, что при создании любого справочника сразу же создаются два индекса:

```
I=CODE  |of CODE  |0  |CODE(UPPER)  |CODE
I=DESCR |of DESCR |0  |DESCR(UPPER) |DESCR
```

То есть, вероятно, именно на использовании этих индексов и базируются методы `ПорядокКодов` и `ПорядокНаименований`.

Для подчиненного справочника `ГородаСтранМира` созданы еще два индекса:

```
I=PCODE  |of PARENT and |0 |PARENTEXT,CODE(UPPER)  |PCODE
I=PDESCR |of PARENT and |0 |PARENTEXT,DESCR(UPPER) |PDESCR
```

Очевидно, что эти индексы ориентированы на поддержку свойства `Владелец` подчиненного справочника.

***Примечание***

Исходный текст этого примера приведен на сопроводительном компакт-диске книги в файле \Part_1\Chapter_04\Topic_4_1\Item_4_1_3\sp_LoadOwnerChildsRef.PAS.

Краткое описание перечисленных методов и свойств справочников системы 1С:Предприятие:

- `ПорядокКодов` (или `OrderByCode`) — устанавливает режим выборки элементов справочника в порядке возрастания кодов элементов. Синтаксис:

  `ПорядокКодов()`

- `ПорядокНаименований` (или `OrderByDescr`) — устанавливает режим выборки элементов справочника в алфавитном порядке наименований элементов. Синтаксис:

  `ПорядокНаименований()`

- `Владелец` (или `Owner`) — предоставляет доступ к значению элемента сопряженного справочника (владельца), которому подчинен выбранный элемент текущего справочника. Синтаксис:

  `Владелец`

На рис. 4.35 представлен результат выполнения программы.

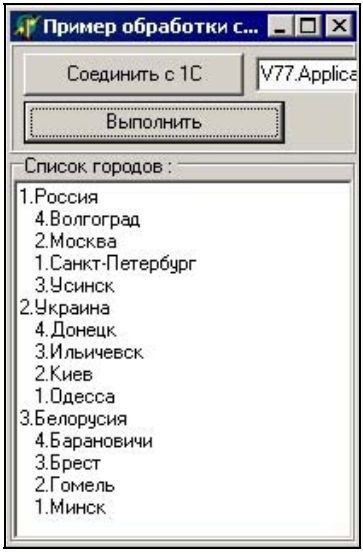



**Рис. 4.35.** Результат работы программы.
Пример изменения порядка сортировки справочников

Кроме выполнения сортировки справочника по свойствам `Код` и `Наименование`, можно воспользоваться методом `ПорядокРеквизита` (или `OrderByAttribute`), позволяющим сортировать выборку элементов справочника по реквизиту.

Этот метод устанавливает режим выборки элементов справочника в порядке возрастания значения указанного реквизита. Его синтаксис:

`ПорядокРеквизита(<ИдентификаторРеквизита>)`

Параметр *<ИдентификаторРеквизита>* — строковое выражение с идентификатором реквизита справочника, задающего порядок обхода элементов справочника.

Данный метод обычно используется до вызова метода `ВыбратьЭлементы`, который фактически открывает выборку. Дальнейшая выборка при помощи метода `ПолучитьЭлемент` будет осуществляться среди элементов текущего справочника в порядке возрастания значения реквизита.

Метод `ПорядокРеквизита` можно использовать только в том случае, если в Конфигураторе при описании данного реквизита установлен признак `Сортировка` (диалоговое окно **Свойства Реквизита** > вкладка **Дополнительные** > флажок **Сортировка**).

Чтобы посмотреть, как это работает, откроем Конфигуратор 1С:Предприятие, внесем в справочник `ФизЛица` новый реквизит `ГородПроживания` типа `Справочник.ГородаСтранМира` (рис. 4.36) и установим для него признак `Сортировка` (рис. 4.37).

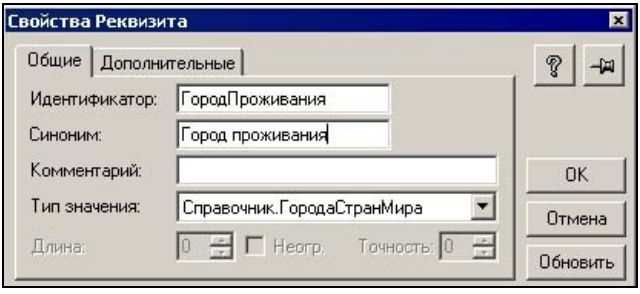



**Рис. 4.36.** Новый реквизит справочника `ФизЛица`

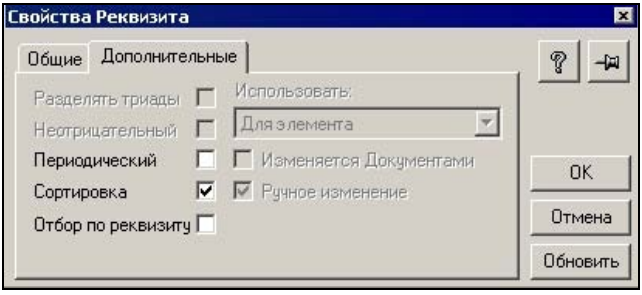



**Рис. 4.37.** Чтобы можно было сортировать справочник по реквизиту, нужно установить для реквизита признак `Сортировка`

С помощью встроенного дизайнера форм программы Конфигуратор 1С:Предприятие нужно внести изменения в форму редактирования элемента справочника `ФизЛица`, как это было сделано ранее (см. рис. 4.17). Это тривиальная процедура, и здесь мы не будем углубляться в детали.

На рис. 4.38 представлен список реквизитов справочника `ФизЛица` после создания нового реквизита `ГородПроживания`.

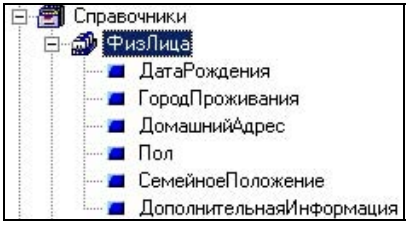



**Рис. 4.38.** Список реквизитов справочника `ФизЛица`

Сохраним сделанные в конфигурации изменения (см. рис. 2.17).

На рис. 4.39 представлена структура справочника `ФизЛица` так, как она отражена в файле 1Cv7.DD. Здесь видно, что в файле SC12.DBF появилось новое поле `SP37` (`ГородПроживания`) и для него создан индекс с именем `VI37` (очевидное следствие установки признака `Сортировка` для этого реквизита).

Чтобы внести информацию о городе проживания для физических лиц, запустим программу 1С:Предприятие, откроем справочник `ФизЛица` и внесем значения реквизита `ГородПроживания` для каждого физического лица (рис. 4.40).

```
#==============================================================================
#==TABLE no 8       : Справочник ФизЛица
# Name     |Descr                          |Type[A/S/U]|DBTableName|ReUsable
T=SC12     |Справочник ФизЛица             |A          |SC12       |1
#-----Fields-------
# Name       |Descr                 |Type|Length|Precision
F=ID         |ID object             |C   |9     |0
F=CODE       |object code           |C   |5     |0
F=DESCR      |object description    |C   |64    |0
F=ISMARK     |Flag Object is Marke  |C   |1     |0
F=VERSTAMP   |Version stamp         |C   |6     |0
F=SP15       |(P)ДатаРождения       |D   |8     |0
F=SP37       |(P)ГородПроживания    |C   |9     |0
F=SP16       |(P)ДомашнийАдрес      |C   |64    |0
F=SP29       |(P)Пол                |C   |9     |0
F=SP30       |(P)СемейноеПоложение  |C   |9     |0
#----Indexes------
# Name      |Descr          |Unique|Indexed fields                          |DBName
I=IDD       |of ID          |0     |ID                                      |IDD
I=CODE      |of CODE        |0     |CODE(UPPER)                             |CODE
I=DESCR     |of DESCR       |0     |DESCR(UPPER)                            |DESCR
I=VI37      |VI37           |0     |SP37                                    |VI37
#
```

**Рис. 4.39.** Содержание файла 1Cv7.DD. Структура справочника `ФизЛица`

| Код | Наименование | Дата... | Город пр... |
|---|---|---|---|
| 5 | Аксакалов Аксакал Аксакалович | 01.01.70 | Киев |
| 1 | Иванов Иван Иванович | 01.01.71 | Москва |
| 6 | Криогенова Фекла Маринадовна | 15.03.65 | Гомель |
| 2 | Петров Петр Петрович | 01.01.72 | Минск |
| 3 | Сидоров Сидор Сидорович | 01.01.73 | Волгоград |
| 4 | Федоров Федор Федорович | 01.01.74 | Киев |

**Рис. 4.40.** Значения реквизита `ГородПроживания` в справочнике `ФизЛица`

Теперь создадим в среде Delphi новый проект. За основу можно взять один из предыдущих примеров (например, см. рис. 4.12). Форма примера представлена на рис. 4.41.

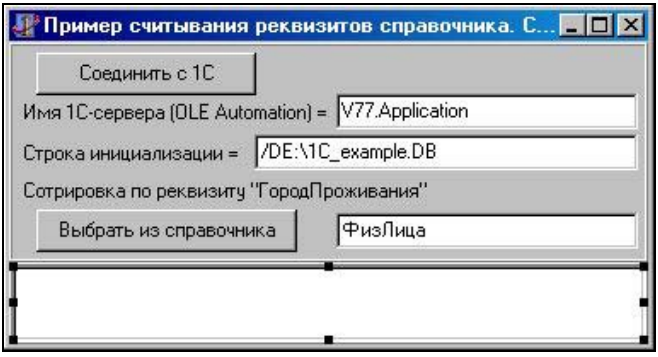



**Рис. 4.41.** Форма примера, иллюстрирующего метод `ПорядокРеквизита` справочника

Изменим обработчик события "нажатие кнопки **Выбрать из справочника**" следующим образом (листинг 4.7).

**Листинг 4.7. Иллюстрация применения метода справочника `ПорядокРеквизита`**

```
procedure TForm1.Button2Click(Sender: TObject);
Var
  OleFizLica: OleVariant;
begin
  ListBox1.Items.Clear;
  if not VarIsEmpty(Ole1C) then begin
     OleFizLica:=Ole1C.CreateObject('Справочник.'+Edit3.Text);
     if not VarIsEmpty(OleFizLica) then begin
        OleFizLica.ПорядокРеквизита('ГородПроживания');
```

```
        if OleFizLica.ВыбратьЭлементы>0 then begin
           while OleFizLica.ПолучитьЭлемент>0 do
            begin
             ListBox1.Items.Add(
                       Trim(OleFizLica.Код)+' | '
                     + Trim(OleFizLica.Наименование)+' |   '
                     + Trim(OleFizLica.ГородПроживания.Владелец.Код)+'.'
          + Trim(OleFizLica.ГородПроживания.Владелец.Наименование)+' | '
                     + Trim(OleFizLica.ГородПроживания.Код)+'.'
                     + Trim(OleFizLica.ГородПроживания.Наименование)
                              );
             Application.ProcessMessages;
           end;
        end;
     end;
  end;
end;
```

Здесь все прозрачно и пояснения не требуются.

***Примечание***

Исходные тексты этого примера находятся на сопроводительном компакт-диске книги в файле \Part_1\Chapter_04\Topic_4_1\Item_4_1_3\sp_LoadRefFromRef.PAS.

Результат выполнения программы представлен на рис. 4.42.

**Рис. 4.42.** Результат выборки из справочника `ФизЛица` с использованием метода `ПорядокРеквизита`

## 4.1.4. Обобщение пройденного материала

Мы достаточно потренировались в программировании на простых и разрозненных примерах. Теперь попробуем все это обобщить и объединить в единое средство, возможности которого будем наращивать по ходу изложения. Назовем его "Простой просмотрщик для 1С:Предприятие (OLE Automation)".

Создадим в среде Delphi новый проект с именем sp_1Cv7_Viewer_via_OLE.DPR.

> ***Примечание***
>
> Исходный текст этого проекта находится на сопроводительном компакт-диске книги в папке \Part_1\Chapter_04\sp_1Cv7_Viewer_via_OLE.PAS\01.

Форма примера представлена на рис. 4.43.

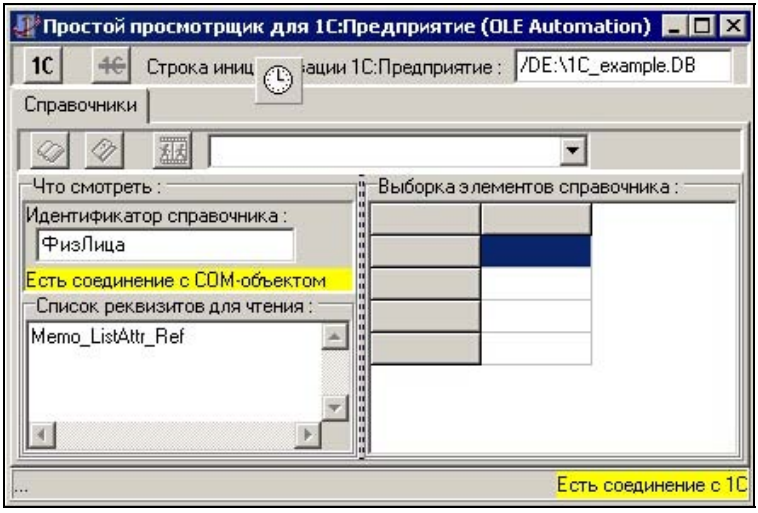



**Рис. 4.43.** Форма просмотрщика для 1С:Предприятие (OLE Automation)

В предыдущих примерах мы обращались к методам, свойствам и реквизитам справочников по их идентификаторам, точно зная, что они актуальны (т. е. присутствуют в конфигурации). В общем случае это недопустимо.

Система 1С:Предприятие (как сервер OLE Automation) предоставляет возможности доступа к метаданным конфигурации, что позволяет проводить анализ информационной структуры из внешних приложений.

Работа с метаданными выполняется через глобальный атрибут `Метаданные` (или `Metadata`), который имеет тип `Метаданные` и всегда присутствует в системе.

Глобальный атрибут `Метаданные` является вспомогательным объектом системы и предназначен для доступа к свойствам структуры метаданных конфигурации. Например, можно проверить, существует ли справочник с заданным идентификатором, используя способ, приведенный в листинге 4.8.

**Листинг 4.8. Проверка существования справочника в заданной конфигурации 1С:Предприятие**

```
Result:=false;
if Ole1C.Метаданные.Справочник(''+idReference).Выбран>0 then begin
   Result:=true;
end;
```

В данном примере переменная `idReference` (типа `string`) содержит тестируемый идентификатор справочника. Пустая строка `''`, присутствующая перед переменной `idReference`, — это не причуды программиста. Система 1С:Предприятие не всегда корректно выполняет вызовы методов, процедур и функций, если не использовать такую "конструкцию".

***Примечание***

Полный текст функции, решающей эту задачу, приведен в файле \sp_Lib\sp_Lib_Delphi.PAS на сопроводительном компакт-диске книги.

Проверить, существует ли реквизит справочника с заданным идентификатором, можно способом, приведенным в листинге 4.9.

**Листинг 4.9. Проверка существования реквизита справочника**

```
Result:=false;
if
Ole1C.Метаданные.Справочник(''+idReference).Реквизит(''+idAttr).Выбран>0
then begin
   Result:=true;
end;
```

В данном примере переменная `idAttr` (типа `string`) содержит идентификатор тестируемого реквизита в справочнике с идентификатором `idReference`.

***Примечание***

Полный текст функции, решающей эту задачу, приведен в файле \sp_Lib\sp_Lib_Delphi.PAS на сопроводительном компакт-диске книги.

У объекта `Метаданные` есть метод `Выбран` (или `Selected`), который проверяет, спозиционирован объект типа `Метаданные` на конкретном объекте метаданных или нет. Синтаксис:

```
Выбран()
```

Метод возвращает число 1 (если объект соответствует объекту метаданных, т. е. спозиционирован) или 0 (если не соответствует). Например, при обращении к массиву подчиненных метаданных по идентификатору, если объекта метаданных с таким идентификатором не существует, возвращается неспозиционированный объект типа `Метаданные`.

Изменим обработчик события "нажатие кнопки **Открыть справочник**" (кнопка с пиктограммой открытой книги, см. рис. 4.43) следующим образом (листинг 4.10).

**Листинг 4.10. Проверка существования справочника при соединении**

```
procedure TForm1.SB_Ref_connectClick(Sender: TObject);
// Создать объект типа "Справочник"
begin
  SB_Ref_disconnect.Click;
  if not VarIsEmpty(ole_1C) then begin
     // Проверка существования справочника в данной конфигурации
     if YesReferenceInThis1C(ole_1C,Trim(Edit_RefName.Text)) then begin
        // Справочник с именем, заданным в Edit_RefName.Text, существует.
        // Создаем этот справочник и получаем на него ссылку
ole_1C_ref:=ole_1C.CreateObject('Справочник.'+Trim(Edit_RefName.Text));
        if not VarIsEmpty(ole_1C_ref) then begin
           //Справочник создан и ссылка на него существует
           Label_YesConnect_to_Obj.Visible:=TRUE;
           SB_Ref_connect.Enabled:=FALSE;
           SB_Ref_disconnect.Enabled:=TRUE;
           SB_Ref_Load.Enabled:=TRUE;
           Edit_RefName.Enabled:=FALSE;
           Edit_RefName.Color:=clSilver;
        end;
     end
     else begin
        ShowMessage('Справочник.'
                  + Trim(Edit_RefName.Text)
```

```
                    + ' не существует в этой конфигурации');
    end;
  end;
end;
```

Поясним текст обработчика.

В данном случае подход тот же, что и ранее (например, в листинге 4.7), единственное отличие — это проверка существования справочника в текущей конфигурации.

> ***Примечание***
>
> Проверку производит функция `YesReferenceInThis1C`, полный текст которой приведен на сопроводительном компакт-диске книги в файле \sp_Lib\sp_Lib_Delphi.PAS.

Если после компиляции и запуска программы ввести идентификатор несуществующего справочника (например, **`ФизЛица_XXX`**), обработчик (листинг 4.10) выведет на экран соответствующее сообщение (рис. 4.44).

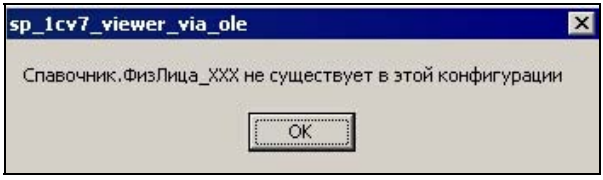



**Рис. 4.44.** Сообщение

Теперь изменим обработчик события "нажатие кнопки **Выполнить действие**" (кнопка с пиктограммой бегущих человечков, см. рис. 4.43) следующим образом (листинг 4.11).

**Листинг 4.11. Обработчик события "нажатие кнопки *Выполнить действие*"**

```
procedure TForm1.SB_Ref_LoadClick(Sender: TObject);
//Сделать выборку элементов справочника
Var
  idAttr: string;  // Идентификатор реквизита справочника
  iCol  : integer; // Номер столбца для SG_Info
begin
  SG_Info_Clear;
  if not VarIsEmpty(ole_1C_ref) then begin
// По умолчанию, если свойство Memo_ListAttr_Ref.Lines
```

```
// не содержит строк, в выборку попадут только
// "Код" и "Наименование" элемента справочника
     if Memo_ListAttr_Ref.Lines.Count>0 then begin
// Если в списке были указаны идентификаторы реквизитов справочника,
// для каждого из них должен быть зарезервирован
// отдельный столбец в SG_Info(типа TStringGrid)
        SG_Info.ColCount:=SG_Info.ColCount+Memo_ListAttr_Ref.Lines.Count;
        for iCol:=2 to (SG_Info.ColCount-1) do
         begin
          idAttr:=Trim(Memo_ListAttr_Ref.Lines[iCol-2]);
          // Идентификаторы реквизитов размещаем в заголовке
          // соответствующего столбца таблицы
          SG_Info.Cells[iCol,0]:=idAttr;
        end;
     end;
     //==========================================
     // Здесь будем в дальнейшем расширять возможности
     // программы для реализации различных спсобов выборки
     // элементов справочника
     if (ComboBox_Action.ItemIndex in [0]) then begin
        // Метод выбора элементов справочника:
        // "ВыбратьЭлементы без учета иерархии"
        //Эта функция реализует непосредственно выборку
        func_Ref_SelectItems_0;
     end;
     //==========================================
  end;
end;
```

Поясним текст обработчика (листинг 4.11). Здесь все тривиально. Проверяется, существует ли ссылка на справочник (переменная `ole_1C_ref`). Затем анализируется список реквизитов для чтения (см. рис. 4.43). Далее, в зависимости от того, какое действие было выбрано (значение `ComboBox_Action.ItemIndex`, рис. 4.45), вызывается соответствующая функция (в данном случае — `func_Ref_SelectItems_0`).

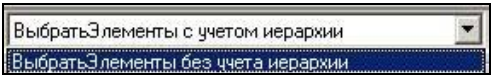



**Рис. 4.45.** Выбор действия при обработке справочника

Текст функции `func_Ref_SelectItems_0` приведен в листинге 4.12.

**Листинг 4.12. Выборка элементов справочника `ФизЛица` без учета иерархии**

```
function TForm1.func_Ref_SelectItems_0: integer;
// Здесь реализован метод выбора элементов "ВыбратьЭлементы" справочника
Var
  idRef,            // Идентификатор справочника,
  idAttr: string;   // идентификатор реквизита,
  sValAttr: string; // значение реквизита,которое нужно
                    // разместить в таблице,
  NrType,           // тип реквизита
  iCol,iRow: integer;
begin
 Result:=0;
 GroupBox_Method_1.Caption:='Метод: ВыбратьЭлементы без учета иерархии';
 if not VarIsEmpty(ole_1C_ref) then begin
    // Считываем идентификатор справочника.
    // Он нужен нам для того, чтобы проверить
    // существование реквизита этого справочника
    idRef:=Trim(Edit_RefName.Text);
    // Выбираем все элементы справочника
    if ole_1C_ref.ВыбратьЭлементы(0)>0 then begin
       // Поочередно извлекаем из выборки
       // все элементы справочника в цикле
       iRow:=0;
       while ole_1C_ref.ПолучитьЭлемент>0 do
        begin
         Result:=Result+1;
         iRow:=iRow+1;
   if iRow>=SG_Info.RowCount then SG_Info.RowCount:=SG_Info.RowCount+1;
         SG_Info.Rows[iRow].Clear;
         // Размещаем в таблице код элемента справочника
         SG_Info.Cells[0,iRow]:=Trim(ole_1C_ref.Код);
         // Размещаем в таблице наименование элемента справочника
         SG_Info.Cells[1,iRow]:=Trim(ole_1C_ref.Наименование);
         if SG_Info.ColCount>2 then begin
            // Если заданы реквизиты для считывания,
            // сканируем соответствующие столбцы и выбираем
```

```
            // из их заголовков идентификаторы реквизитов
            for iCol:=2 to (SG_Info.ColCount-1) do
             begin
              // Считываем идентификатор реквизита
              idAttr:=Trim(SG_Info.Cells[iCol,0]);
              // Определяем тип реквизита
              NrType:=GetTypeAttrReference_1C(ole_1C,idRef,idAttr);
              sValAttr:= '<???>';
        //==========================================================
        // Считываем реквизит с учетом его типа
              if (NrType in [1..3]) then begin // Базовый тип
        //Строка, Число или Дата
                 sValAttr:=Trim(ole_1C_ref.ПолучитьАтрибут(''+idAttr));
              end;
              if NrType=10 then begin // Перечисление
  sValAttr:= Trim(ole_1C_ref.ПолучитьАтрибут(''+idAttr).Идентификатор);
              end;
              if NrType=11 then begin // Справочник
  sValAttr:= Trim(ole_1C_ref.ПолучитьАтрибут(''+idAttr).Наименование);
              end;
        //==========================================================
        // Записываем считанное значение реквизита
        // в соответствующий столбец
              SG_Info.Cells[iCol,iRow]:=sValAttr;
            end;
        end;
        Application.ProcessMessages;
      end;
    end;
 end;
end;
```

Поясним текст обработчика (листинг 4.12).

Как и раньше (например, см. листинг 4.7), выборка из справочника производится с использованием метода `ВыбратьЭлементы`, а позиционирование на каждом элементе справочника — с использованием метода `ПолучитьЭлемент`.

Затем в обязательном порядке считываются `Код` и `Наименование` элемента справочника.

Далее, если указаны реквизиты, то в цикле считываются их идентификаторы, определяется тип реквизита (функция `GetTypeAttrReference_1C`) и, в зависимости от типа реквизита, считывается его свойство (как значение для размещения в таблице).

> ***Примечание***
>
> Полный текст функции `GetTypeAttrReference_1C` приведен в файле \sp_Lib\sp_Lib_Delphi.PAS на сопроводительном компакт-диске книги.

Чтобы проверить, как все это заработает, скомпилируем проект и запустим его на выполнение. Укажем справочник `ФизЛица` и внесем в список реквизитов для чтения идентификаторы соответствующих реквизитов (рис. 4.46).

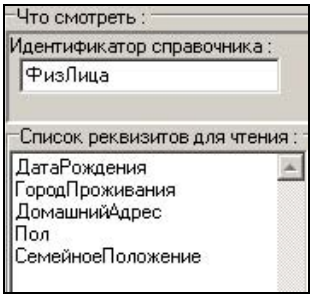



**Рис. 4.46.** Список реквизитов справочника, значения которых нужно выбрать

Теперь запустим 1С:Предприятие (как сервер OLE Automation), нажав кнопку **1С** (см. рис. 4.43). Затем откроем справочник (кнопка с пиктограммой открытой книги) и запустим выборку элементов справочника (кнопка с пиктограммой бегущих человечков). Результат представлен на рис. 4.47.

Выборка элементов справочника :

| Код | Наименование | ДатаРожде | ГородПрожи | ДомашнийАдрес | Пол | СемейноеПоложение |
|---|---|---|---|---|---|---|
| 5 | Аксакалов Аксакал Аксакалов | 01.01.1970 | Киев | Россия, г.Б, ул.Б, д.00 | Мужской | Разведен |
| 1 | Иванов Иван Иванович | 01.01.1971 | Москва | Россия, г.Неизвестный, ул.А, д.01 | Мужской | СостоитВЗарегистрированномБраке |
| 6 | Криогенова Фекла Маринадо | 15.03.1965 | Гомель | Россия, г.N, ул., У, д. 005 | Женский | НикогдаНеСостоялВБраке |
| 2 | Петров Петр Петрович | 01.01.1972 | Минск | Россия, г.Неизвестный, ул.А, д.02 | Мужской | ГражданскийБрак |
| 3 | Сидоров Сидор Сидорович | 01.01.1973 | Волгоград | Россия, г.Неизвестный, ул.А, д.03 | Мужской | Вдовец |
| 4 | Федоров Федор Федорович | 01.01.1974 | Киев | Россия, г.Известный, ул.Б., д.001 | Мужской | Разошелся |

**Рис. 4.47.** Результат выборки элементов справочника

Теперь попробуем прочитать элементы справочника с учетом иерархии. Для начала — с учетом владельца. Текст функции, решающей эту задачу, приведен в листинге 4.13.

**Листинг 4.13. Выборка элементов справочника ФизЛица с учетом владельца**

```
function TForm1.func_Ref_SelectItems_1: integer;
// Здесь реализован метод выбора элементов
// справочника "ВыбратьЭлементы" с учетом владельца
Var
  ole_1CRef_Owner,   // Ссылка на справочник-владелец
  ole_1CRef_OwnerTekEl: OleVariant; // Ссылка на текущий элемент
                                    // справочника-владельца
  idRefOwner,
  idRef,            // Идентификатор справочника,
  idAttr: string;   // идентификатор реквизита,
  sValAttr: string; // значение реквизита,которое нужно
                    // разместить в таблице,
  NrType,           // тип реквизита
  iCol,iRow: integer;
begin
 Result:=0;
 ole_1CRef_Owner:=UnAssigned;
 ole_1CRef_OwnerTekEl:=UnAssigned;
 GroupBox_Method_1.Caption:='Метод: ВыбратьЭлементы с учетом владельца;
 if not VarIsEmpty(ole_1C_ref) then begin
    // Считываем идентификатор справочника.
    // Он нам нужен для того, чтобы проверить существование
    // реквизита этого справочника
    idRef:=Trim(Edit_RefName.Text);
    // Проверяем, есть ли владелец у этого справочника
if ole_1C.Метаданные.Справочник(''+idRef).Владелец.Выбран>0 then begin
// Справочник-владелец существует
// Считываем идентификатор справочника-владельца
idRefOwner:=Trim(ole_1C.Метаданные.Справочник(''+idRef).Владелец.
  Идентификатор);
//Создаем объект: справочник-владелец
        ole_1CRef_Owner:=ole_1C.CreateObject('Справочник.'+idRefOwner);
        if not VarIsEmpty(ole_1CRef_Owner) then begin
// Выбираем владельца
        if ole_1CRef_Owner.Выбрать('Выберите владельца','')>0 then begin
// Читаем ссылку на текущий элемент справочника
              ole_1CRef_OwnerTekEl:=ole_1CRef_Owner.ТекущийЭлемент;
```

```
            end;
         end;
     end;
     if not VarIsEmpty(ole_1CRef_OwnerTekEl) then begin
// Если владелец был выбран, вызываем соответствующий
// метод для фильтрации
        ole_1C_ref.ИспользоватьВладельца(ole_1CRef_OwnerTekEl,0);
// В некоторых случаях вызов метода, как в предыдущей строке,
// может не сработать. Тогда возможен следующий вариант:
//ole_1C_ref.ИспользоватьВладельца(ole_1CRef_Owner.ТекущийЭлемент,0);
     end;
// Выбираем все элементы справочника с учетом иерархии
     if ole_1C_ref.ВыбратьЭлементы(1)>0 then begin
// Поочередно извлекаем из выборки
// все элементы справочника в цикле
        iRow:=0;
        while ole_1C_ref.ПолучитьЭлемент>0 do
         begin
          Result:=Result+1;
          iRow:=iRow+1;
   if iRow>=SG_Info.RowCount then SG_Info.RowCount:=SG_Info.RowCount+1;
          SG_Info.Rows[iRow].Clear;
// Размещаем в таблице код элемента справочника
          SG_Info.Cells[0,iRow]:=Trim(ole_1C_ref.Код);
// Размещаем в таблице наименование элемента справочника
          SG_Info.Cells[1,iRow]:=Trim(ole_1C_ref.Наименование);
          if SG_Info.ColCount>2 then begin
// Если заданы реквизиты для считывания, сканируем соответствующие
// столбцы и выбираем из их заголовков идентификаторы реквизитов
            for iCol:=2 to (SG_Info.ColCount-1) do
              begin
          // Считываем идентификатор реквизита
               idAttr:=Trim(SG_Info.Cells[iCol,0]);
          // Определяем тип реквизита
               NrType:=GetTypeAttrReference_1C(ole_1C,idRef,idAttr);
               sValAttr:= '<???>';
//=========================================================
// Считываем реквизит с учетом его типа
               if (NrType in [1..3]) then begin // Базовый тип
```

```
        //Строка, Число или Дата
              sValAttr:=Trim(ole_1C_ref.ПолучитьАтрибут(''+idAttr));
            end;
            if NrType=10 then begin // Перечисление
  sValAttr:= Trim(ole_1C_ref.ПолучитьАтрибут(''+idAttr).Идентификатор);
            end;
            if NrType=11 then begin // Справочник
  sValAttr:= Trim(ole_1C_ref.ПолучитьАтрибут(''+idAttr).Наименование);
            end;
//=========================================================
// Записываем считанное значение реквизита в соответствующий столбец
            SG_Info.Cells[iCol,iRow]:=sValAttr;
          end;
        end;
        Application.ProcessMessages;
      end;
    end;
 end;
end;
```

Поясним текст обработчика (листинг 4.13).

Здесь в строке:

```
if ole_1C.Метаданные.Справочник(''+idRef).Владелец.Выбран>0 then begin
```

определяется, есть ли у данного справочника (`idRef`) справочник-владелец. Если нет, все происходит так же, как и в функции `func_Ref_SelectItems_0` (листинг 4.12). Если справочник-владелец существует, то создаем ссылку на этот справочник:

```
ole_1CRef_Owner:=ole_1C.CreateObject('Справочник.'+idRefOwner);
```

Затем вызываем метод справочника-владельца для выбора элемента в окне диалога:

```
if ole_1CRef_Owner.Выбрать('Выберите владельца','')>0 then begin
```

При вызове этого метода на экране появится диалоговое окно 1С:Предприятие (рис. 4.49), где можно выбрать двойным щелчком элемент справочника. Если элемент справочника был выбран, считываем ссылку на него с помощью свойства `ТекущийЭлемент` справочника-владельца:

```
ole_1CRef_OwnerTekEl:=ole_1CRef_Owner.ТекущийЭлемент;
```

Если ссылка на `ТекущийЭлемент` корректна, вызываем метод `ИспользоватьВладельца` подчиненного справочника:

```
ole_1C_ref.ИспользоватьВладельца(ole_1CRef_OwnerTekEl,0);
```

или

```
ole_1C_ref.ИспользоватьВладельца(ole_1CRef_Owner.ТекущийЭлемент,0);
```

Тем самым обеспечивается фильтрация выборки элементов справочника при вызове метода `ВыбратьЭлементы`.

Установить выборку подчиненного справочника позволяет метод `ИспользоватьВладельца` (или `UseOwner`). Синтаксис:

```
ИспользоватьВладельца(Владелец, ФлагИзменения)
```

Параметры: *<Владелец>* (необязательный) — выражение со значением элемента справочника-владельца; *<ФлагИзменения>* (необязательный) — регулирует возможность интерактивного изменения владельца (1 — пользователь может изменить владельца интерактивно, 0 — пользователь не может интерактивно изменить владельца). Метод возвращает значение элемента справочника-владельца для текущего подчиненного справочника (на момент до исполнения метода).

Добавим в свойство `Lines` компонента `ComboBox_Action` (Delphi) строку:

```
"ВыбратьЭлементы с учетом владельца"
```

Изменим обработчик события "нажатие кнопки **Выполнить действие** (`SB_Ref_Load`)" следующим образом (листинг 4.14).

**Листинг 4.14. Обработчик события "нажатие кнопки *Выполнить действие*"**

```
//=========================================
if (ComboBox_Action.ItemIndex in [0]) then begin
...
end;
if (ComboBox_Action.ItemIndex in [1]) then begin
    // Метод выбора элементов справочника:
    // "ВыбратьЭлементы с учетом иерархии"
    // Эта функция реализует непосредственно выборку
    func_Ref_SelectItems_1;
end;
//=========================================
```

И запустим наш проект на выполнение.

Создадим соединение с 1С:Предприятие.

Укажем идентификатор справочника, имеющего владельца, способ выборки (с помощью раскрывающегося списка) и запустим выборку элементов справочника (рис. 4.48).

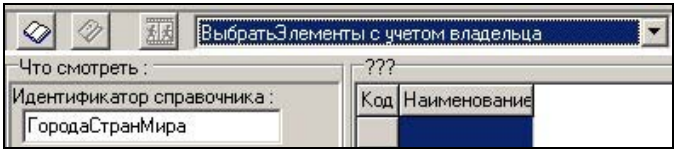



**Рис. 4.48.** Режим выборки элементов справочника

В ответ на запрос (вызов метода `Выбрать` справочника) программа 1С:Предприятие выведет на экран диалоговое окно выбора элементов справочника (рис. 4.49).

**Рис. 4.49.** Диалоговое окно (1С:Предприятие)
для интерактивного выбора элемента справочника

Выбор элемента производится двойным щелчком мыши на соответствующей строке.

***Примечание***

Окно программы 1С:Предприятие (как сервера OLE Automation) по умолчанию остается невидимым. Поэтому диалоговое окно, показанное на рис. 4.49, может быть скрыто другими окнами различных приложений, запущенных в данный момент на компьютере. В таком случае можно "найти" диалоговое окно с помощью сочетания клавиш <Alt>+<Tab>. На рис. 4.50 показана пиктограмма, которую нужно выбрать для того, чтобы диалоговое окно появилось поверх остальных окон.

**Рис. 4.50.** Пиктограмма,
соответствующая приложению 1С:Предприятие

На рис. 4.51 представлен результат выборки элементов справочника `ГородаСтранМира` для элемента справочника-владельца `Россия`.

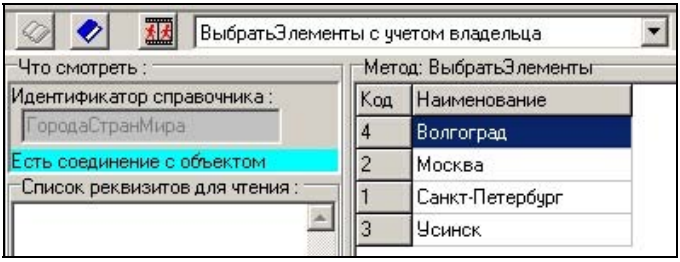



**Рис. 4.51.** Результат выборки из справочника.
Фильтрация с применением метода `ИспользоватьВладельца`

Объекты типа `Справочник` в системе 1С:Предприятие могут иметь иерархическую структуру. Это значит, что элементы справочников могут быть сгруппированы определенным образом. Чтобы посмотреть, как механизм OLE Automation позволяет делать это, вновь изменим структуру справочника `ФизЛица`. Для этого запустим Конфигуратор 1С:Предприятие и откроем справочник `ФизЛица` (см. рис. 2.13).

Внесем изменения, показанные на рис. 4.52.

**Рис. 4.52.** Установка глубины вложенности
иерархической структуры справочника `ФизЛица`

Сохраним конфигурацию (см. рис. 2.17). Созданные нами изменения отразятся в словаре данных (файл 1Cv7.DD, рис. 4.53).

Как видите, в файле SC12.DBF появились два дополнительных поля `PARENTID` и `ISFOLDER`, а также индекс `VIP37`.

Теперь запустим программу 1С:Предприятие (см. рис. 2.23) и откроем справочник `ФизЛица`. В списке справочника откроем всплывающее меню (рис. 4.54) и выберем пункт **Иерархический список**.

Затем опять откроем всплывающее меню и выберем пункт **Новая группа** (рис. 4.55).

На экране появится диалоговое окно редактирования группы справочника, где нужно внести наименование группы и нажать кнопку **OK** (рис. 4.56).

Далее последует запрос подтверждения изменений (рис. 4.57).

Результат представлен на рис. 4.58.

```
#==============================================================================
#==TABLE no 8      : Справочник ФизЛица
# Name     |Descr                          |Type[A/S/U]|DBTableName|ReUsable
T=SC12     |Справочник ФизЛица             |A          |SC12       |1
#-----Fields-------
# Name       |Descr                |Type|Length|Precision
F=ID         |ID object            |C   |9     |0
F=PARENTID   |ID parent obj        |C   |9     |0
F=CODE       |object code          |C   |5     |0
F=DESCR      |object description   |C   |64    |0
F=ISFOLDER   |Flag - Is Line - Fol|N   |1     |0
F=ISMARK     |Flag Object is Marke|C   |1     |0
F=VERSTAMP   |Version stamp        |C   |6     |0
F=SP15       |(P)ДатаРождения      |D   |8     |0
F=SP37       |(P)ГородПроживания   |C   |9     |0
F=SP16       |(P)ДомашнийАдрес     |C   |64    |0
F=SP29       |(P)Пол               |C   |9     |0
F=SP30       |(P)СемейноеПоложение|C   |9     |0
#----Indexes------
# Name      |Descr          |Unique|Indexed fields                                          |DBName
I=IDD       |of ID          |0     |ID                                                      |IDD
I=PCODE     |of PARENT and  |0     |PARENTID,ISFOLDER,CODE(UPPER)                           |PCODE
I=PDESCR    |of PARENT and  |0     |PARENTID,ISFOLDER,DESCR(UPPER)                          |PDESCR
I=CODE      |of CODE        |0     |CODE(UPPER)                                             |CODE
I=DESCR     |of DESCR       |0     |DESCR(UPPER)                                            |DESCR
I=VI37      |VI37           |0     |SP37                                                    |VI37
I=VIP37     |VIP37          |0     |PARENTID,ISFOLDER,SP37                                  |VIP37
#
```

**Рис. 4.53.** Содержание файла 1Cv7.DD. Структура справочника `ФизЛица`

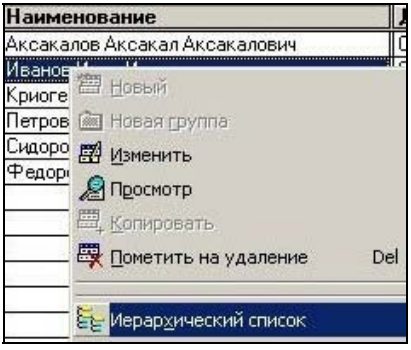



**Рис. 4.54.** Включение/отключение режима визуализации справочника `ФизЛица` с учетом иерархической структуры

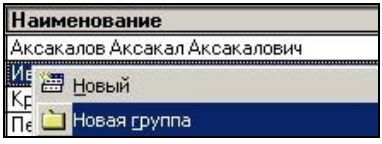



**Рис. 4.55.** Новая группа в справочнике

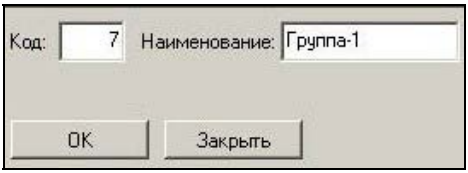



**Рис. 4.56.** Диалоговое окно редактирования группы справочника

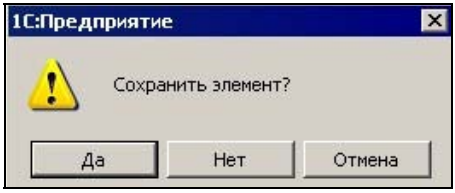



**Рис. 4.57.** Запрос на сохранение изменений в справочнике

**Рис. 4.58.** Список справочника в режиме отображения иерархии

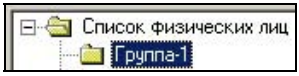



**Рис. 4.59.** Выбор группы в иерархическом списке справочника

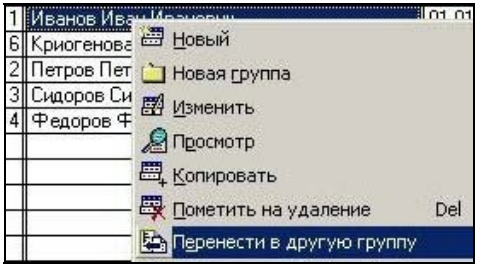



**Рис. 4.60.** Перенос элемента справочника в другую группу

Как видите, группа создана, но в эту группу не входит ни один из элементов справочника. Чтобы перенести элемент справочника в группу, отметим соответствующую группу (рис. 4.59).

Затем отметим тот элемент справочника, который нужно перенести в группу, и откроем всплывающее меню, где выберем пункт **Перенести в другую группу** (рис. 4.60).

Последует запрос подтверждения переноса (рис. 4.61).

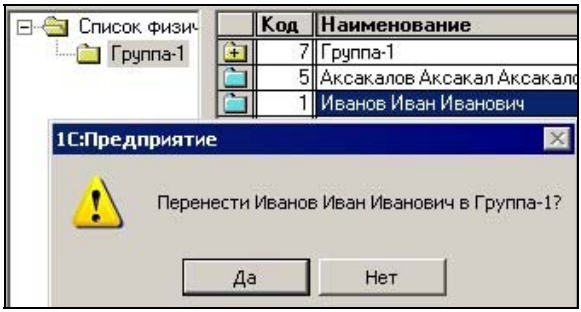



**Рис. 4.61.** Запрос подтверждения переноса элемента справочника в другую группу

На рис. 4.62 представлен результат, когда все физические лица (кроме элемента с кодом 6) разнесены по группам.

На рис. 4.63 представлено содержание группы "Группа-1".

**Рис. 4.62.** Список справочника в режиме отображения иерархии

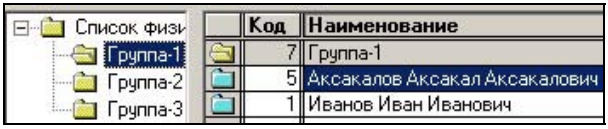



**Рис. 4.63.** Список справочника в режиме отображения иерархии. "Группа-1" справочника

На рис. 4.64 представлен список справочника, когда отключен режим иерархического отображения.

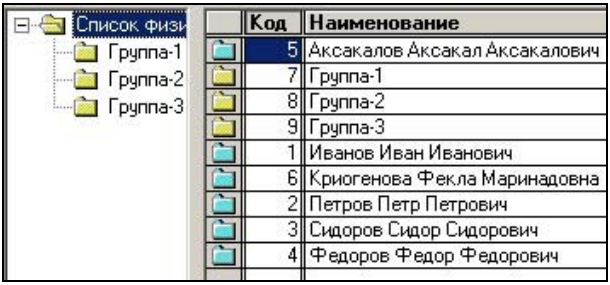



**Рис. 4.64.** Список справочника при выключенном режиме отображения иерархии

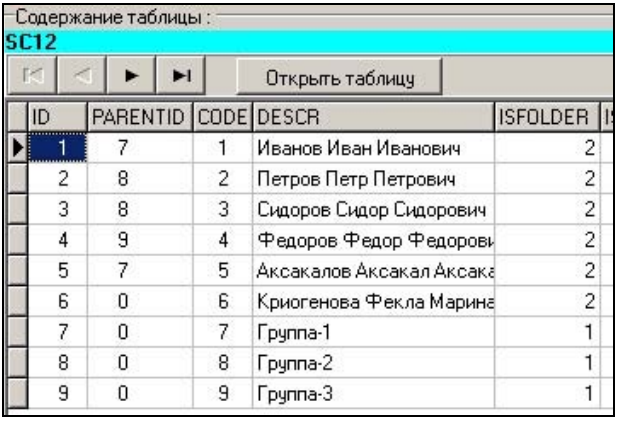



**Рис. 4.65.** Содержание файла SC12.DBF

Откроем файл SC12.DBF в любом просмотрщике DBF-файлов (рис. 4.65).

Здесь видно, что поле `PARENTID` содержит ссылку на соответствующую группу справочника.

Теперь вернемся к нашему проекту (Delphi). Создадим новую форму диалога с именем `Dlg_Sel_Grp_In_Ref` (рис. 4.66).

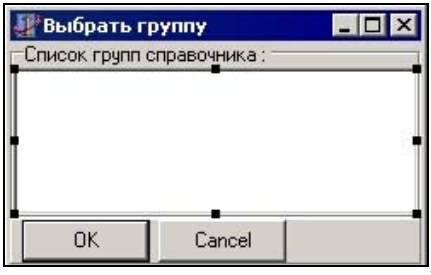



**Рис. 4.66.** Форма диалога для выбора группы справочника

Создадим метод `Execute`, который должен обеспечить выбор группы справочника (листинг 4.15).

**Листинг 4.15. Метод `Execute` формы диалога для выбора группы в справочнике**

```
function TDlg_Sel_Grp_In_Ref.Execute(ole_1C_ref: OleVariant): string;
// Метод возвращает код выбранной группы.
// ole_1C_ref - корректная ссылка на справочник.
Var
  Sx: string;
begin
  Result:='';
  ListBox1.Items.Clear;
  if not VarIsEmpty(ole_1C_ref) then begin
     if ole_1C_ref.ВыбратьЭлементы(0)>0 then begin
        while ole_1C_ref.ПолучитьЭлемент>0 do
         begin
           if ole_1C_ref.ЭтоГруппа>0 then begin
              // Выбираем только группы в справочнике
              Sx:=Trim(ole_1C_ref.Код);
              Sx:=Sx+'.'+Trim(ole_1C_ref.Наименование);
              ListBox1.Items.ADD(Sx);
           end;
        end;
     end;
     if ShowModal = mrOk then begin
        if ListBox1.ItemIndex>=0 then begin
           // Если группа выбрана, выделим из строки код группы
           Sx:=ListBox1.Items[ListBox1.ItemIndex];
           Result:=Trim(sp_GetWordFromString('.',Sx,1));
        end;
     end;
  end;
end;
```

Поясним текст обработчика (листинг 4.15).

Здесь в цикле отбираются только группы справочника (для проверки используется метод справочника `ЭтоГруппа`). Таким образом, заполняется список `ListBox1`.

Если выбор в списке произведен и нажата кнопка **OK**, из выбранной строки выделяется код группы (это делает функция `sp_GetWordFromString`).

> ***Примечание***
>
> Полный исходный код этой функции приведен на сопроводительном компакт-диске книги в файле \sp_Lib\sp_Lib_Delphi.PAS.

Создадим функцию, реализующую вывод информации с учетом группировки (листинг 4.16).

**Листинг 4.16. Выборка элементов справочника `ФизЛица` с учетом родителя**

```
function TForm1.func_Ref_SelectItems_2: integer;
// Здесь реализован метод выбора элементов справочника:
// ВыбратьЭлементы с учетом родителя
Var
  ole_1CRef_ParentTekEl: OleVariant; // Ссылка на группу
                                     // (текущий элемент)
  idRefParent,    // Идентификатор группы
  idRef,          // Идентификатор справочника
  idAttr: string; // Идентификатор реквизита
  sValAttr: string; // Значение реквизита, которое
                    // нужно разместить в таблице
  NrType, // Тип реквизита
  NrMode, // Режим открытия выборки
  OleBuf: oleVariant;
  iCol,iRow: integer;
begin
 Result:=0;
 ole_1CRef_ParentTekEl:=UnAssigned;
 idRefParent:='';
 GroupBox_Method_1.Caption:='Метод: ВыбратьЭлементы (с учетом родителя)';
 if not VarIsEmpty(ole_1C_ref) then begin
    // Считываем идентификатор справочника.
    // Он нам нужен для того, чтобы проверить
    // существование реквизита этого справочника
    idRef:=Trim(Edit_RefName.Text);
    // По умолчанию устанавливаем режим выборки
    // элементов справочника без учета иерархии
    NrMode:=0;
```

```
   // Проверяем, иерархический ли это справочник
   if ole_1C.Метаданные.Справочник(''+idRef).КоличествоУровней>1 then begin
      // Это иерархический справочник.
      // Устанавливаем режим выборки элементов справочника
      // с учетом иерархии
      NrMode:=1;
      // Выбираем родителя (группу)
      idRefParent:=Dlg_Sel_Grp_In_Ref.Execute(ole_1C_ref);
      if length(idRefParent)>0 then begin
         // Если выбрали родителя — позиционируемся на нем
         if (ole_1C_ref.НайтиПоКоду(''+idRefParent,0))*1>0 then begin
            // Установим фильтрацию по выбранному родителю (группе)
            ole_1CRef_ParentTekEl:=ole_1C_ref.ТекущийЭлемент;
            ole_1C_ref.ИспользоватьРодителя(ole_1CRef_ParentTekEl,0);
            // В некоторых случаях вызов метода,
            // как в предыдущей строке, может не сработать.
            //Тогда возможен следующий вариант:
         //ole_1C_ref.ИспользоватьРодителя(ole_1C_ref.ТекущийЭлемент,0);
         end;
      end;
   end;
   // Выбираем все элементы справочника с учетом иерархии
   if ole_1C_ref.ВыбратьЭлементы(NrMode*1)>0 then begin
      // Далее, поочередно извлекаем из выборки
      // все элементы справочника в цикле
      iRow:=0;
      while ole_1C_ref.ПолучитьЭлемент>0 do
       begin
        Result:=Result+1;
        iRow:=iRow+1;
 if iRow>=SG_Info.RowCount then SG_Info.RowCount:=SG_Info.RowCount+1;
        SG_Info.Rows[iRow].Clear;
        // Размещаем в таблице Код элемента справочника
        SG_Info.Cells[0,iRow]:=Trim(ole_1C_ref.Код);
        // Размещаем в таблице наименование элемента справочника
        SG_Info.Cells[1,iRow]:=Trim(ole_1C_ref.Наименование);
        if SG_Info.ColCount>2 then begin
           // Если заданы реквизиты для считывания,
           // сканируем соответствующие столбцы и выбираем
```

```
            // из их заголовков идентификаторы реквизитов
            for iCol:=2 to (SG_Info.ColCount-1) do
             begin
              // Считываем идентификатор реквизита
              idAttr:=Trim(SG_Info.Cells[iCol,0]);
              // ОПРЕДЕЛЯЕМ тип реквизита
              NrType:=GetTypeAttrReference_1C(ole_1C,idRef,idAttr);
              sValAttr:= '<???>';
//=========================================================
              // Считываем реквизит с учетом его типа
              if (NrType in [1..3]) then begin // Базовый тип
              // Строка, Число или Дата
                 sValAttr:=Trim(ole_1C_ref.ПолучитьАтрибут(''+idAttr));
              end;
              if NrType=10 then begin // Перечисление
  sValAttr:= Trim(ole_1C_ref.ПолучитьАтрибут(''+idAttr).Идентификатор);
              end;
              if NrType=11 then begin // Справочник
  sValAttr:= Trim(ole_1C_ref.ПолучитьАтрибут(''+idAttr).Наименование);
              end;
              if NrType=12 then begin // Документ
                 oleBuf:= ole_1C_ref.ПолучитьАтрибут(''+idAttr);
                 sValAttr:= Trim(oleBuf.Вид)
                           + '  N '
                           + Trim(oleBuf.НомерДок)
                           + ' ['
                           + Trim(oleBuf.ДатаДок)
                           + ']';
              end;
//=========================================================
// Записываем считанное значение реквизита в соответствующий столбец
              SG_Info.Cells[iCol,iRow]:=sValAttr;
            end;
         end;
         Application.ProcessMessages;
       end;
    end;
 end;
end;
```

Функция func_Ref_SelectItems_2 позволяет проверить, является ли данный справочник иерархическим. Для этого используется объект Метаданные. Если справочник имеет иерархическую структуру, то атрибут КоличествоУровней принимает значение больше 1. Если справочник иерархический, то вызывается метод Dlg_Sel_Grp_In_Ref.Execute (листинг 4.15), который возвращает код выбранной группы. Далее используется метод справочника НайтиПоКоду для того, чтобы получить значение свойства выбранной группы ТекущийЭлемент. Затем вызывается метод справочника ИспользоватьРодителя, который и определяет фильтрацию элементов справочника при выборке.

> ***Примечание***
>
> Обратите внимание: умножение на 1 — это не каприз программиста, а нюансы работы системы 1С:Предприятие посредством механизма OLE Automation.

Далее производится выборка элементов справочника и визуализация информации, в частности, реквизитов типа Документ.

Метод ИспользоватьРодителя (или UseParent) справочника позволяет установить выборку по группе справочника. Синтаксис:

ИспользоватьРодителя(<*Группа*>)

Необязательный параметр <*Группа*> — выражение со значением группы справочника. Метод возвращает значение текущей группы для справочника (на момент до исполнения метода).

Данный метод используется до вызова метода ВыбратьЭлементы, который фактически открывает выборку. Дальнейшая выборка при помощи метода ПолучитьЭлемент будет выполняться только среди элементов текущего справочника, принадлежащих указанной группе.

Добавим в свойство Lines компонента ComboBox_Action (Delphi) строку:

"ВыбратьЭлементы с учетом родителя"

Изменим обработчик события "нажатие кнопки **Выполнить действие** (SB_Ref_Load)" следующим образом (листинг 4.10).

**Листинг 4.17. Обработчик события "нажатие кнопки *Выполнить действие*"**

```
//=========================================
if (ComboBox_Action.ItemIndex in [0]) then begin
...
end;
if (ComboBox_Action.ItemIndex in [1]) then begin
...
```

```
end;
if (ComboBox_Action.ItemIndex in [2]) then begin
   // Метод выбора элементов справочника:
   // "ВыбратьЭлементы с учетом Родителя"
   // Эта функция реализует непосредственно выборку
   func_Ref_SelectItems_2;
end;
//=========================================
```

И запустим наш проект на выполнение.

1. Создадим соединение с 1С:Предприятие.
2. Укажем идентификатор справочника с иерархической структурой, способ выборки (с помощью раскрывающегося списка) и запустим выборку элементов справочника (рис. 4.67).

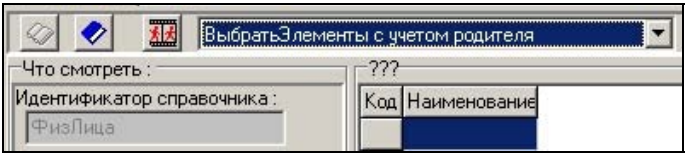



**Рис. 4.67.** Выбор элементов справочника с учетом родителя

Если справочник иерархический, на экран будет выведено диалоговое окно для выбора группы (рис. 4.68).

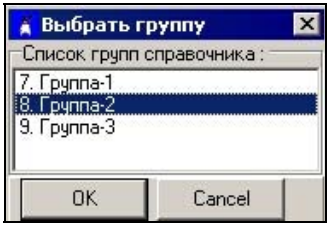



**Рис. 4.68.** Выбор группы справочника

На рис. 4.69 показан результат выборки для справочника `ФизЛица` по группе с кодом 8 ("Группа-2").

На рис. 4.70 показан результат выборки элементов справочника `ФизЛица` для случая, когда группа не указана (см. рис. 4.68).

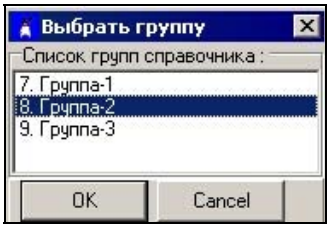



**Рис. 4.69.** Результат выборки элементов справочника о заданной группе

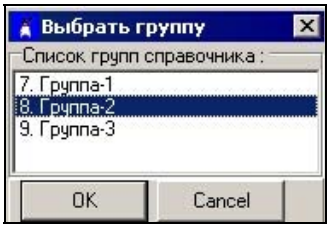



**Рис. 4.70.** Результат выборки из справочника с иерархической структурой

В этом случае метод `ИспользоватьРодителя` не вызывается (листинг 4.16), но методу `ВыбратьЭлементы` в качестве параметра передается 1, что означает "выбирать элементы с учетом иерархии".

Как видите, при выводе результатов выборки порядок иерархии соблюдается.

## 4.1.5. Периодические реквизиты справочников

*Периодические реквизиты* — это реквизиты, значения которых связаны с датой. При изменении значения периодического реквизита старое значение сохраняется, при этом новое значение действует с указанной даты, а старое — до указанной даты. Работа с периодическими реквизитами осуществляется при помощи методов `Получить` и `ИспользоватьДату`, а также при помощи специального агрегатного типа данных `Периодический` (или `Periodic`).

Вновь доработаем нашу конфигурацию. Внесем изменения в структуру справочника `ФизЛица`. Для этого запустим Конфигуратор 1С и создадим новое перечисление `СостояниеФизлица` (рис. 4.71).

Далее, откроем окно редактирования свойств справочника `ФизЛица` (см. рис. 2.13) и внесем новый реквизит `СостояниеФизлица` (рис. 4.72).

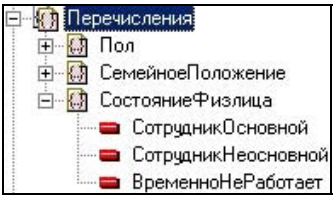



**Рис. 4.71.** Содержание перечисления `СостояниеФизлица`

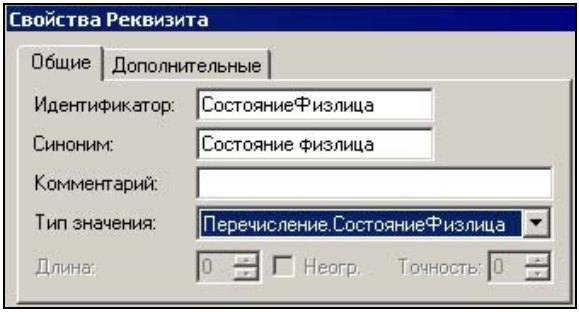



**Рис. 4.72.** Свойства реквизита справочника `СостояниеФизлица`

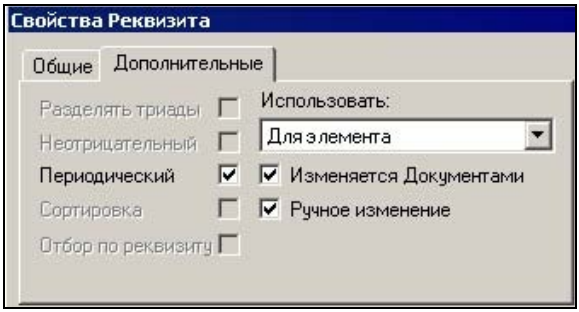



**Рис. 4.73.** Признак "периодичности" реквизита справочника

Перейдя на вкладку **Дополнительные**, укажем, что этот реквизит будет периодическим (рис. 4.73).

***Примечание***

Более подробную информацию о свойствах периодических реквизитов системы 1С:Предприятие можно найти в специальной литературе.

В результате список реквизитов справочника `ФизЛица` будет иметь вид, показанный на рис. 4.74.

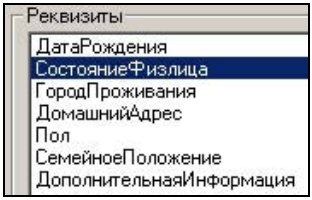



**Рис. 4.74.** Список реквизитов в справочнике `ФизЛица`

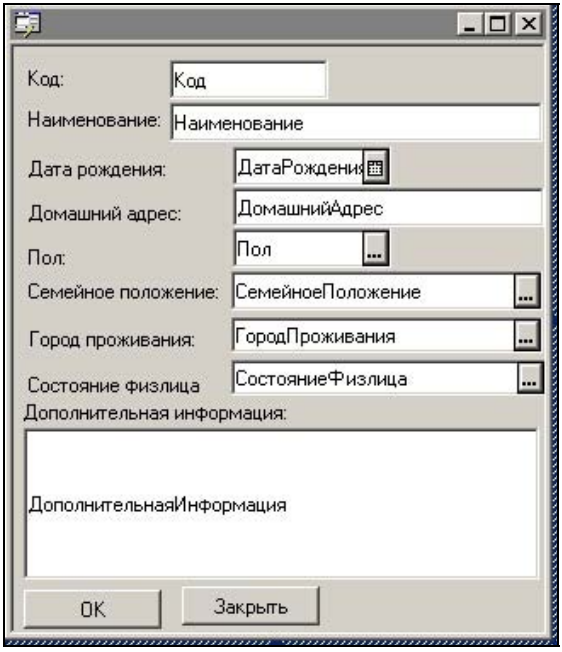



**Рис. 4.75.** Форма редактирования элементов справочника `ФизЛица`

Внесем соответствующие изменения в форму редактирования элемента справочника `ФизЛица` (рис. 4.75).

После этого сохраним изменения в конфигурации (см. рис. 2.17).

***Примечание***

Если открыть словарь данных (файл 1Cv7.DD), то можно увидеть, что содержание файла не изменилось. То есть несмотря на то, что в справочник `ФизЛица` был внесен новый реквизит, это не было отражено в словаре данных.

Запустим программу 1С:Предприятие, откроем справочник `ФизЛица` и сделаем двойной щелчок на строке с кодом 4. На экране появится форма редактирования элемента справочника (рис. 4.76).

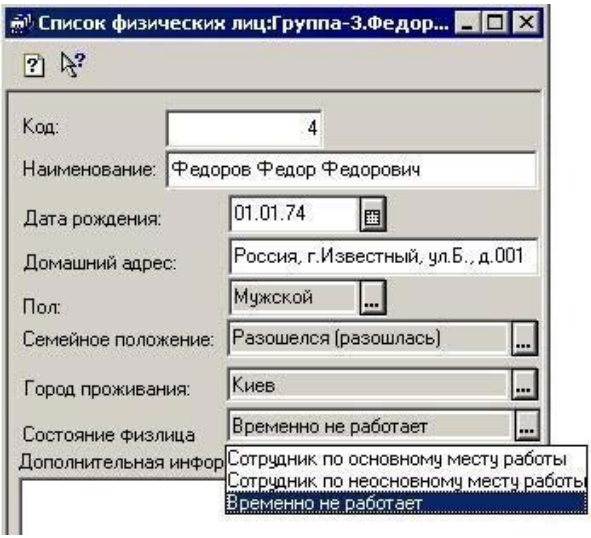



**Рис. 4.76.** Редактирование элемента справочника в диалоге

Выберем значение реквизита "Состояние физлица" в раскрывающемся списке **Состояние физлица** (см. рис. 4.76).

После закрытия формы редактирования (нажатием кнопки **ОК**) и подтверждения в ответ на запрос о сохранении на экране появится диалоговое окно, предлагающее подтвердить сохранение изменений периодических реквизитов элемента справочника (рис. 4.77).

***Примечание***

Значения периодических реквизитов устанавливаются на определенную (рабочую) дату. Рабочую дату можно установить в диалоговом окне настройки системы (рис. 4.80).

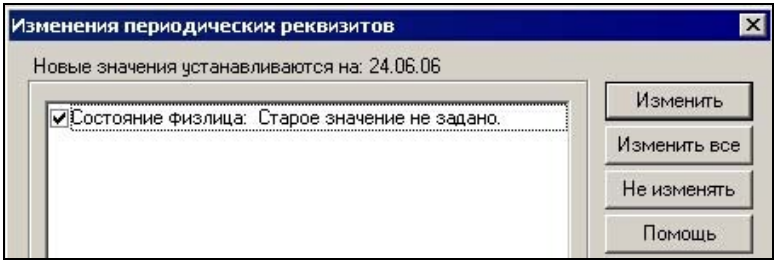



**Рис. 4.77.** Запрос на подтверждение сохранения значений периодических реквизитов элемента справочника

Далее, точно так же заполним значения реквизита "Состояние физлица" для остальных элементов справочника. Результат приведен на рис. 4.78.

| Код | Наименование | Дата ... | Состояние физлица |
|---|---|---|---|
| 5 | Аксакалов Аксакал Аксак | 01.01.70 | Сотрудник по неосновному месту работы |
| 7 | Группа-1 | | |
| 8 | Группа-2 | | |
| 9 | Группа-3 | | |
| 1 | Иванов Иван Иванович | 01.01.71 | Сотрудник по основному месту работы |
| 6 | Криогенова Фекла Марин | 15.03.65 | Сотрудник по основному месту работы |
| 2 | Петров Петр Петрович | 01.01.72 | Временно не работает |
| 3 | Сидоров Сидор Сидорович | 01.01.73 | Сотрудник по неосновному месту работы |
| 4 | Федоров Федор Федоров | 01.01.74 | Временно не работает |

**Рис. 4.78.** Список элементов справочника `ФизЛица`

Для того чтобы можно было экспериментировать с периодическими реквизитами, нужно для каждого элемента справочника внести еще значения реквизита "Состояние физлица", но на другую дату.

Выберем в меню **Сервис** программы 1С:Предприятие пункт **Параметры** (рис. 4.79).

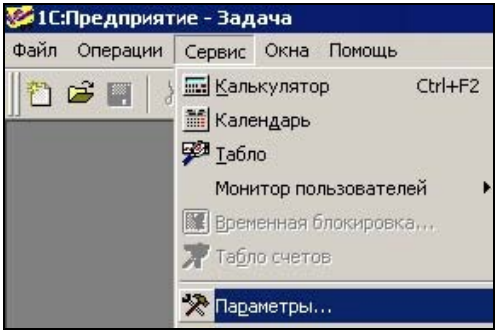



**Рис. 4.79.** Выбор пункта меню **Сервис** для настройки системы 1С:Предприятие

На экране появится диалоговое окно настройки системы 1С:Предприятие (рис. 4.80).

На вкладке **Общие** этого окна установим новую рабочую дату. Затем внесем значения периодического реквизита "Состояние физлица" на новую дату. Результат представлен на рис. 4.81.

В табл. 4.4 приведены значения периодического реквизита `СостояниеФизлица` на даты "07.06.2006" и "24.06.2006".

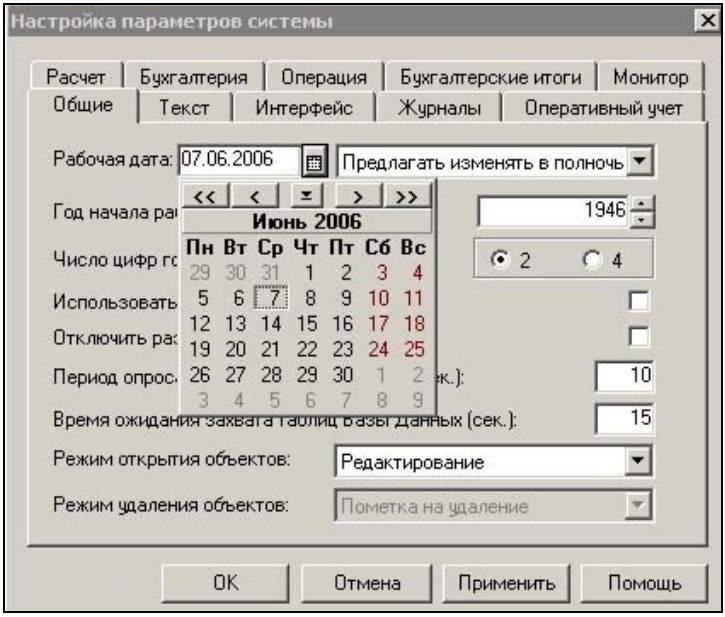



**Рис. 4.80.** Диалоговое окно настройки системы 1С:Предприятие

| | Код | Наименование | Дата... | Состояние физлица |
|---|---|---|---|---|
| | 5 | Аксакалов Аксакал Аксакалов | 01.01.70 | Временно не работает |
| | 7 | Группа-1 | | |
| | 8 | Группа-2 | | |
| | 9 | Группа-3 | | |
| | 1 | Иванов Иван Иванович | 01.01.71 | Сотрудник по неосновному месту работы |
| | 6 | Криогенова Фекла Маринадов | 15.03.65 | Временно не работает |
| | 2 | Петров Петр Петрович | 01.01.72 | Сотрудник по основному месту работы |
| | 3 | Сидоров Сидор Сидорович | 01.01.73 | Сотрудник по основному месту работы |
| | 4 | Федоров Федор Федорович | 01.01.74 | Сотрудник по основному месту работы |

**Рис. 4.81.** Список элементов справочника `ФизЛица`

***Таблица 4.4.*** *Значения периодического реквизита* `СостояниеФизлица`

| Код | Наименование | Состояние на 07.06.2006 | Состояние на 24.06.2006 |
|---|---|---|---|
| 1 | Иванов Иван Иванович | Сотрудник по неосновному месту работы | Сотрудник по основному месту работы |
| 2 | Петров Петр Петрович | Сотрудник по основному месту работы | Временно не работает |
| 3 | Сидоров Сидор Сидорович | Сотрудник по основному месту работы | Сотрудник по неосновному месту работы |

***Таблица 4.4*** *(окончание)*

| Код | Наименование | Состояние на 07.06.2006 | Состояние на 24.06.2006 |
|---|---|---|---|
| 4 | Федоров Федор Федорович | Сотрудник по основному месту работы | Временно не работает |
| 5 | Аксакалов Аксакал Аксакалович | Временно не работает | Сотрудник по неосновному месту работы |
| 6 | Криогенова Фекла Маринадовна | Временно не работает | Сотрудник по основному месту работы |

Для того чтобы понять, как именно система 1С:Предприятие размещает значения периодических реквизитов, с помощью любого просмотрщика DBF-файлов откроем файл SC12.DBF и убедимся, что его содержание не изменилось. Затем откроем файл 1SCONST.DBF (рис. 4.82).

Содержание таблицы :
1SCONST.DBF

Открыть таблицу

| OBJID | ID | DATE | PARTNO | VALUE |
|---|---|---|---|---|
| 5 | 16 | 24.06.2006 | 0 | 14 |
| 1 | 16 | 24.06.2006 | 0 | 13 |
| 4 | 16 | 24.06.2006 | 0 | 15 |
| 6 | 16 | 24.06.2006 | 0 | 13 |
| 2 | 16 | 24.06.2006 | 0 | 15 |
| 3 | 16 | 24.06.2006 | 0 | 14 |
| 5 | 16 | 07.06.2006 | 0 | 15 |
| 1 | 16 | 07.06.2006 | 0 | 14 |
| 2 | 16 | 07.06.2006 | 0 | 13 |
| 3 | 16 | 07.06.2006 | 0 | 13 |
| 4 | 16 | 07.06.2006 | 0 | 13 |
| 6 | 16 | 07.06.2006 | 0 | 15 |

**Рис. 4.82.** Содержание файла 1SCONST.DBF

***Примечание***

*До* этого момента файл 1SCONST.DBF был пустым. То есть очевидно, что значения периодических реквизитов справочников система 1С:Предприятие размещает в файле 1SCONST.DBF.

Вернемся к нашему проекту (Delphi) и сделаем изменения, которые обеспечивали бы возможность считывания значений периодических реквизитов справочников.

Добавим в свойство `Lines` компонента `ComboBox_Action` (Delphi) строку:

`"ВыбратьЭлементы с учетом родителя и периодических реквизитов"`

Создадим функцию, реализующую вывод значений периодических реквизитов на заданную дату (поскольку текст этой функции имеет значительную длину, здесь он не приводится).

> ***Примечание***
>
> Полный текст этой функции находится на сопроводительном компакт-диске книги в папке \Part_1\Chapter_04\sp_1Cv7_Viewer_via_OLE.PAS\01. Здесь приведено лишь то, что является существенным с точки зрения считывания значений периодических реквизитов.

Следует отметить, что в данном случае речь идет о считывании значений на заданную дату. В листинге 4.18 приведена часть текста функции `func_Ref_SelectItems_3`.

**Листинг 4.18. Считывание значений периодических реквизитов на заданную дату**

```
function TForm1.func_Ref_SelectItems_3: integer;
// В этой функции учтена возможность считывания
// значений периодических реквизитов
Var
...
  // Дата, на которую считываются периодические реквизиты:
  sDate_for_AttrPeriod: string;
begin
...
 sDate_for_AttrPeriod:='24.06.2006';
 if InputQuery('Дата для пер.рекв.','Введите дату:',sDate_for_AttrPeriod)
 then begin
    sDate_for_AttrPeriod:=Trim(sDate_for_AttrPeriod);
    if length(sDate_for_AttrPeriod)>0 then begin
       ole_1C_ref.ИспользоватьДату(''+sDate_for_AttrPeriod,1);
    end;
    ...
    ...
 end;
end;
```

Поясним текст функции (здесь все достаточно просто).

После ввода корректной даты (`InputQuery`) вызывается метод `Использовать-Дату` справочника, где в качестве параметров передаются введенная дата (строка) и единица. В остальном считывание значений периодических реквизитов (на заданную дату) ничем не отличается от того, как это делалось ранее.

Установить дату выборки периодических реквизитов справочника позволяет метод `ИспользоватьДату` (или `UseDate`) объекта `Периодический`. Синтаксис:

`ИспользоватьДату(<Дата>, <УстСразу>)`

Параметры: *<Дата>* — выражение со значением типа `Дата`; *<УстСразу>* (необязательный) — число (1 — дата, переданная в качестве параметра, будет установлена уже в текущей выборке; 0 — дата, переданная в качестве параметра, будет установлена при следующем открытии выборки; значение по умолчанию — 0).

Изменим обработчик `SB_Ref_Load` события "нажатие кнопки **Выполнить действие**" следующим образом (листинг 4.19).

**Листинг 4.19. Обработчик события "нажатие кнопки *Выполнить действие*"**

```
//=========================================
if (ComboBox_Action.ItemIndex in [0]) then begin
...
end;
if (ComboBox_Action.ItemIndex in [1]) then begin
...
end;
if (ComboBox_Action.ItemIndex in [2]) then begin
...
end;
if (ComboBox_Action.ItemIndex in [3]) then begin
//Считывание периодических реквизитов
   func_Ref_SelectItems_3;
end;
//=========================================
```

Скомпилируем и запустим наш проект на выполнение. Создадим соединение с 1С:Предприятие. Укажем идентификатор справочника, в структуру которого входят периодические реквизиты, способ выборки (в раскрывающемся списке) и запустим выборку элементов справочника (рис. 4.83).

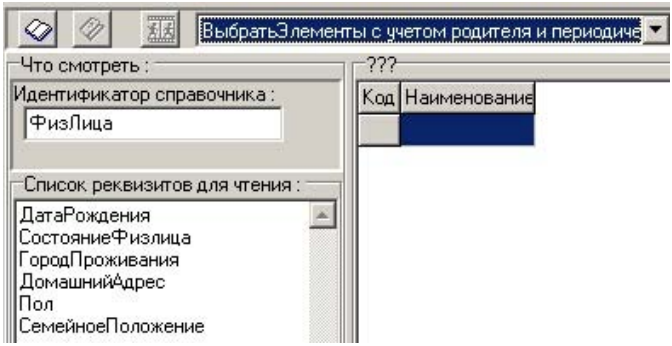



**Рис. 4.83.** Выборка элементов справочника и чтение значений реквизита, включая периодические

На экране появится диалоговое окно ввода даты, на которую нужно прочитать значения периодических реквизитов (рис. 4.84).

Результат считывания представлен на рис. 4.85.

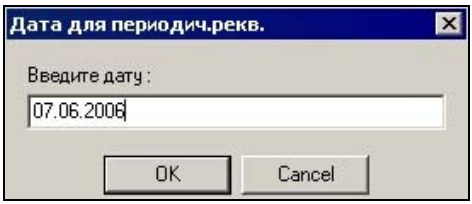



**Рис. 4.84.** Ввод даты, на которую нужно прочитать значения периодических реквизитов

Метод: ВыбратьЭлементы (с учетом родителя и периодич.рекв.)

| Код | Наименование | ДатаРожде | СостояниеФизлица | ГородПрож | ДомашнийАдре |
|---|---|---|---|---|---|
| 7 | Группа-1 | | | | |
| 5 | Аксакалов Аксакал Аксакалович | 01.01.1970 | ВременноНеРаботает | Киев | Россия, г.Б, ул. |
| 1 | Иванов Иван Иванович | 01.01.1971 | СотрудникНеосновной | Москва | Россия, г.Неиз |
| 8 | Группа-2 | | | | |
| 2 | Петров Петр Петрович | 01.01.1972 | СотрудникОсновной | Минск | Россия, г.Неиз |
| 3 | Сидоров Сидор Сидорович | 01.01.1973 | СотрудникОсновной | Волгоград | Россия, г.Неиз |
| 9 | Группа-3 | | | | |
| 4 | Федоров Федор Федорович | 01.01.1974 | СотрудникОсновной | Киев | Россия, г.Изве |
| 6 | Криогенова Фекла Маринадовна | 15.03.1965 | ВременноНеРаботает | Гомель | Россия, г.N, ул. |

**Рис. 4.85.** Результат считывания значений периодических реквизитов на 07.06.2006

Вновь запустим выборку элементов справочника, но укажем дату "24.06.2006". Результат считывания представлен на рис. 4.86.

Метод: ВыбратьЭлементы (с учетом родителя и периодич.рекв.)

| Код | Наименование | ДатаРожде | СостояниеФизлица | ГородПрож | ДомашнийАдр |
|---|---|---|---|---|---|
| 7 | Группа-1 | | | | |
| 5 | Аксакалов Аксакал Аксакалович | 01.01.1970 | СотрудникНеосновной | Киев | Россия, г.Б, ул |
| 1 | Иванов Иван Иванович | 01.01.1971 | СотрудникОсновной | Москва | Россия, г.Неиз |
| 8 | Группа-2 | | | | |
| 2 | Петров Петр Петрович | 01.01.1972 | ВременноНеРаботает | Минск | Россия, г.Неиз |
| 3 | Сидоров Сидор Сидорович | 01.01.1973 | СотрудникНеосновной | Волгоград | Россия, г.Неиз |
| 9 | Группа-3 | | | | |
| 4 | Федоров Федор Федорович | 01.01.1974 | ВременноНеРаботает | Киев | Россия, г.Изве |
| 6 | Криогенова Фекла Маринадовна | 15.03.1965 | СотрудникОсновной | Гомель | Россия, г.N, ул |

**Рис. 4.86.** Результат считывания значений периодических реквизитов на 24.06.2006

Результат считывания можно сопоставить с той информацией, что мы вводили в программе 1С:Предприятие (табл. 4.4).

Точно так же считываются значения периодических реквизитов, имеющих базовый тип (например, тип `Число`). Чтобы проверить это, внесем в справочник `ФизЛица` периодический реквизит `КолвоДетей` (тип `Число`) и заполним элементы справочника соответствующими значениями. На рис. 4.87 и 4.88 представлены результаты выборки из справочника.

Часто требуется прочитать историю изменений значения периодического реквизита справочника. Для того чтобы проиллюстрировать это, доработаем наш проект. Создадим новую форму (рис. 4.89).

ВыбратьЭлементы с учетом родителя и периодиче

Что смотреть :

Идентификатор спр

ФизЛица

Есть соединение с

Список реквизитов

ДатаРождения
КолвоДетей
СостояниеФиз.

Метод: ВыбратьЭлементы (с учетом родителя и периодич.рекв.)

| Код | Наименование | ДатаРожде | КолвоДетей | СостояниеФизлица |
|---|---|---|---|---|
| 7 | Группа-1 | | | |
| 5 | Аксакалов Аксакал Аксакалович | 01.01.1970 | 3 | ВременноНеРаботает |
| 1 | Иванов Иван Иванович | 01.01.1971 | 2 | СотрудникНеосновной |
| 8 | Группа-2 | | | |
| 2 | Петров Петр Петрович | 01.01.1972 | 1 | СотрудникОсновной |
| 3 | Сидоров Сидор Сидорович | 01.01.1973 | 1 | СотрудникОсновной |
| 9 | Группа-3 | | | |
| 4 | Федоров Федор Федорович | 01.01.1974 | 2 | СотрудникОсновной |
| 6 | Криогенова Фекла Маринадовна | 15.03.1965 | 0 | ВременноНеРаботает |

**Рис. 4.87.** Результат считывания значений периодических реквизитов на 01.05.2006

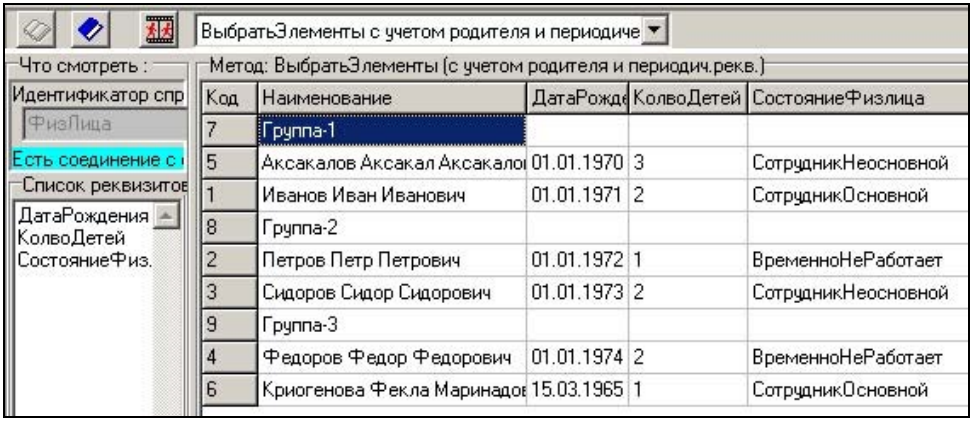



**Рис. 4.88.** Результат считывания значений периодических реквизитов на 24.06.2006

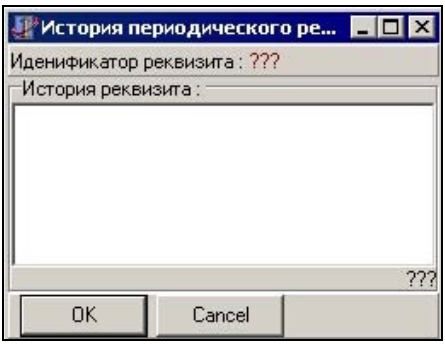



**Рис. 4.89.** Форма (Delphi) для визуализации истории изменения значений периодического реквизита

Внесем изменения в текст, сгенерированной Delphi для этой формы по умолчанию (листинг 4.20).

**Листинг 4.20. Текст, сгенерированный Delphi для формы `TDlg_AttrPeriod_History`**

```
type
  TDlg_AttrPeriod_History = class(TForm)
    Panel1: TPanel;
    OKBtn: TButton;
    CancelBtn: TButton;
    Panel3: TPanel;
    Label2: TLabel;
```

```
    Label_id_Attr_Period: TLabel;
    GroupBox1: TGroupBox;
    ListBox1: TListBox;
    Label_id_Ref: TLabel;  private
    { Private declarations }
    ole_Ref: OleVariant;   // Ссылка на объект типа "Справочник"
    ole_TekEl: OleVariant; // Ссылка на текущий элемент справочника
  public
    { Public declarations }
    function Execute(idRef,idAttrPeriod,sCode: string): boolean;
  end;
```

Как видите, у формы появились 2 новых свойства:

- `ole_Ref` — ссылка на объект типа `Справочник`;
- `ole_TekEl` — ссылка на текущий элемент справочника.

> ***Примечание***
>
> Термин "элемент справочника" в системе 1С:Предприятие адекватен понятию "строка таблицы базы данных". Значения всех периодических реквизитов всех справочников содержатся в отдельной таблице (файл 1SCONST.DBF, см. рис. 4.12) и, следовательно, "привязаны" к элементам справочников (по принципу Master-Detail). Более подробно об этом рассказано в *части II*.

Для того чтобы прочитать историю изменения значений реквизита справочника с использованием механизма OLE Automation, нужно спозиционировать справочник на нужном элементе и получить на него ссылку ("текущий элемент").

Получить значение элемента справочника позволяет метод `ТекущийЭлемент` (или `CurrentItem`) справочника. Синтаксис:

`ТекущийЭлемент()`

Метод `ТекущийЭлемент` возвращает значение элемента справочника в целом как объект. Данный метод применяется, например, если нужно передать элемент справочника как параметр в вызове какого-либо метода или присвоить какому-либо реквизиту.

В листинге 4.21 представлен текст метода `Execute` формы (рис. 4.89).

**Листинг 4.21. Метод `Execute` формы (просмотр истории периодического реквизита)**

```
function TDlg_AttrPeriod_
History.Execute(idRef,idAttrPeriod,sCode:string):boolean;
// Входные параметры:
```

```
// idRef — идентификатор справочника,
// idAttrPeriod — идентификатор периодического реквизита,
// sCode — код строки справочника.
Var
 ole_period: OleVariant; // Ссылка на объект типа "Периодический"
 sValAttr: string;
 NrType: integer;
begin
 Result:=FALSE;
 ole_period:=UnAssigned;
 ole_Ref:=UnAssigned;
 ole_TekEl:=UnAssigned;
 ListBox1.Items.Clear;
 idAttrPeriod:=Trim(idAttrPeriod);
 idRef:=Trim(idRef);
 sCode:=Trim(sCode);
 Label_id_Attr_Period.Caption:='';
 Label_id_Ref.Caption:='';
 if (length(idAttrPeriod)>0) and (length(sCode)>0) then begin
   Label_id_Attr_Period.Caption:=idAttrPeriod;
   ole_Ref:=Form1.ole_1C.CreateObject('Справочник.'+idRef);
   if not VarIsEmpty(ole_Ref) then begin
     if (ole_Ref.НайтиПоКоду(''+sCode,0))*1>0 then begin
       ole_TekEl:=ole_Ref.ТекущийЭлемент;
       // Проверяем, есть ли ссылка на "ТекущийЭлемент" справочника
       if not VarIsEmpty(ole_TekEl) then begin
         // Метод "Вид" текущего элемента справочника возвращает
         // идентификатор справочника
         Label_id_Ref.Caption:= Trim(ole_TekEl.Вид);
         // Считываем тип реквизита
         sValAttr:= Label_id_Ref.Caption; // Чтобы укоротить строку
                                          // в книге
NrType:=GetTypeAttrReference_1C(Form1.ole_1C,sValAttr,idAttrPeriod);
         // Создаем объект "Периодический" для того,
         // чтобы прочитать историю значений периодического
         // реквизита справочника
         ole_period:=Form1.ole_1C.CreateObject('Периодический');
         if not VarIsEmpty(ole_period) then begin
```

```
if ole_period.ИспользоватьОбъект(''+idAttrPeriod, ole_TekEl)*1>0
  then begin
             ole_period.ВыбратьЗначения;
             while ole_period.ПолучитьЗначение>0 do
              begin
              sValAttr:= '<???>';
//=========================================================
// Считываем реквизит с учетом его типа
              if (NrType in [1..3]) then begin // Базовый тип
                // Строка, Число или Дата
                sValAttr:= Trim(ole_period.Значение);
              end;
              if NrType=10 then begin // Перечисление
                sValAttr:= Trim(ole_period.Значение.Идентификатор);
              end;
              if NrType=11 then begin // Справочник
                sValAttr:= Trim(ole_period.Наименование);
              end;
//=========================================================
// Результат считывания записываем в ListBox1
      ListBox1.Items.ADD(Trim(ole_period.ДатаЗнач)+' | '+sValAttr);
             end;
           end
           else begin
             ShowMessage('Неудачный вызов метода ИспользоватьОбъект()');
           end;
         end
         else begin
      ShowMessage('Неудачное создание объекта типа "Периодический"');
         end;
         if ShowModal=mrOk then begin
            Result:=TRUE;
         end;
       end;
     end;
   end;
 end;
 ole_period:=UnAssigned;
```

```
 ole_TekEl:=UnAssigned;
 ole_Ref:=UnAssigned;
 ListBox1.Items.Clear;
end;
```

Поясним листинг 4.21. Как видите, в качестве входных параметров методу `Execute` передаются:

- `idRef` — идентификатор справочника;
- `idAttrPeriod` — идентификатор периодического реквизита (историю изменения значений которого нужно прочитать);
- `sCode` — код строки справочника (для позиционирования на соответствующем элементе справочника).

Далее выполняются следующие действия:

1. Создание объекта типа `Справочник` с идентификатором `idRef` в контексте уже открытого сервера OLE Automation 1С:Предприятие (`Form1.ole_1C`) и получение ссылки на него:

   ```
   ole_Ref:=Form1.ole_1C.CreateObject('Справочник.'+idRef);
   ```

2. Позиционирование справочника на элементе с кодом `sCode`:

   ```
   if (ole_Ref.НайтиПоКоду(''+sCode,0))*1>0 then begin
   ```

3. Получение ссылки на текущий элемент справочника (соответствует строке таблицы SC12.DBF с кодом `sCode`):

   ```
   ole_TekEl:=ole_Ref.ТекущийЭлемент;
   ```

4. Создание объекта типа `Периодический` в контексте уже открытого сервера OLE Automation 1С:Предприятие (`Form1.ole_1C`) и получение ссылки на него:

   ```
   ole_period:=Form1.ole_1C.CreateObject('Периодический');
   ```

5. Вызов метода `ИспользоватьОбъект` для созданного объекта типа `Периодический`. В качестве параметров передаются идентификатор периодического реквизита и ссылка на текущий элемент справочника:

   ```
   if ole_period.ИспользоватьОбъект(''+idAttrPeriod, ole_TekEl)*1>0
     then
   ```

   Очевидно, что это определение фильтра по таблице (файлу) 1SCONST.DBF.

6. Выбор значений периодического реквизита (фильтруем строки таблицы 1SCONST.DBF):

   ```
   ole_period.ВыбратьЗначения;
   ```

7. Поочередный (в цикле) выбор строк из полученной выборки:
   `while ole_period.`**`ПолучитьЗначение`**`>0 do`
8. Далее, в зависимости от типа реквизита, считывание соответствующих строковых значений реквизита (более подробно это рассмотрено ранее).

Задать объект применения позволяет метод `ИспользоватьОбъект` (или `UseObject`) объекта `Периодический`. Синтаксис:

`ИспользоватьОбъект(<`*`ИмяРеквизита`*`>, <`*`Объект`*`>)`

Параметры: *<ИмяРеквизита>* — строковое выражение, задающее идентификатор периодического реквизита справочника, как указано в конфигураторе; *<Объект>* — значение элемента справочника, для которого задается применение объекта `Периодический`. Возвращаемое значение (число): 1 — если вызов метода закончился успешно, 0 — если нет.

Метод `ИспользоватьОбъект` задает соответствие созданного ранее объекта `Периодический` тому периодическому реквизиту справочника, для которого он будет применяться. Если параметр *<ИмяРеквизита>* не задан (пустая строка), а параметр *<Объект>* задает элемент справочника, то выборка будет осуществляться по всем периодическим реквизитам справочника.

Открыть выборку периодических значений в интервале дат позволяет метод `ВыбратьЗначения` (или `Selectltems`) объекта `Периодический`. Синтаксис:

`ВыбратьЗначения(<`*`ДатаНачала`*`>, <`*`ДатаКонца`*`>)`

Параметры: *<ДатаНачала>* (необязательный) — выражение типа `Дата` (дата начала периода выборки периодических значений; если параметр не задан, то выборка начинается с самой ранней имеющейся даты); *<ДатаКонца>* (необязательный) — выражение типа `Дата` (дата конца периода выборки периодических значений; если параметр не задан, то выборка заканчивается самой последней имеющейся датой). Возвращаемое значение (число): 1 — если вызов метода закончился успешно, 0 — если нет.

Метод `ВыбратьЗначения` предоставляет возможность (открывает выборку) выбирать периодические значения при помощи метода `ПолучитьЗначение`. Дальнейшая выборка при помощи метода `ПолучитьЗначение` будет происходить среди периодических значений текущего объекта применения, заданного методом `ИспользоватьОбъект`.

Получить из выборки следующее периодическое значение позволяет метод `ПолучитьЗначение` (или `GetValue`) объекта `Периодический`. Синтаксис:

`ПолучитьЗначение()`

Возвращаемое значение (число): 1 — если элемент выбран успешно, 0 — если элемент не выбран (отсутствует).

Метод `ПолучитьЗначение` выбирает периодическое значение в последовательности выборки, открытой перед этим при помощи метода `ВыбратьЗначения` или `ВыбратьПоДокументу`. Само полученное значение следует считывать из атрибута `Значение`. Дату этого значения можно считать из атрибута `ДатаЗнач`. Если значение периодического реквизита установлено документом, то этот документ можно получить при помощи метода `ТекущийДокумент`. Метод `ТекущийОбъект` позволит определить значение текущего элемента справочника, а метод `ТекущийРеквизит` — определить наименование текущего реквизита справочника.

Теперь разместим в главной форме приложения компонент `PopupMenu1`, присвоим свойству `PopupMenu` компонента `SG_Info` значение `PopupMenu1` и создадим пункт меню "История периодического реквизита" (рис. 4.90).

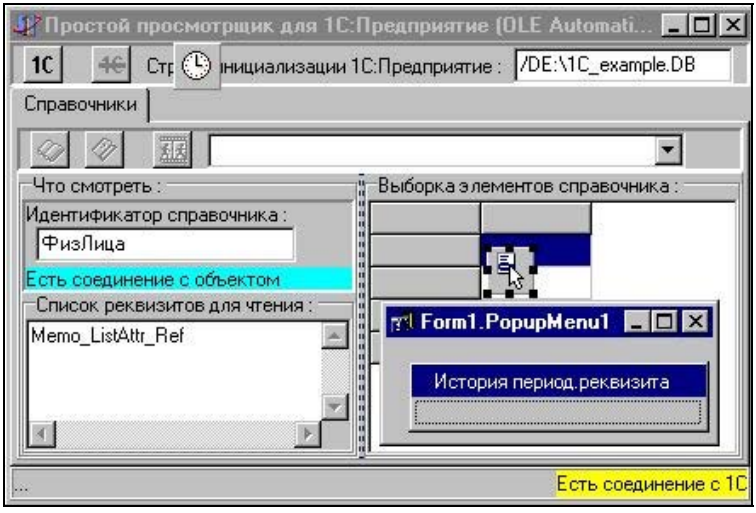



**Рис. 4.90.** Настройка всплывающего меню
для вызова формы "История периодического реквизита"

Создадим обработчик события `OnClick` для пункта меню "История периодического реквизита" (листинг 4.22).

**Листинг 4.22. Обработчик события `OnClick` пункта меню "История периодического реквизита"**

```
procedure TForm1.N1Click(Sender: TObject);
Var
 idRef, // Идентификатор справочника
```

```
 idAttr,// Идентификатор периодического реквизита
 sCode: string; // Код строки справочника
 oleBuf: OleVariant; // Вспомогательная переменная
begin
  if not VarIsEmpty(ole_1C_ref) then begin
     if (SG_Info.Row>0) and (SG_Info.Col>=2) then begin
        sCode:=Trim(SG_Info.Cells[0,SG_Info.Row]);
        if length(sCode)>0 then begin
           idRef:=Trim(Edit_RefName.Text);
           // Идентификатор реквизита, который указал Пользователь
           idAttr:=Trim(SG_Info.Cells[SG_Info.Col,0]);
           // Проверяем, существует ли реквизит idAttr
           // в справочнике idRef
           if YesAttrInReference_1C(ole_1C,idRef,idAttr) then begin
              // Проверяем, является ли реквизит idAttr
              // в справочнике idRef периодическим
              oleBuf:=ole_1C.Метаданные.Справочник(''+idRef);
           if (oleBuf.Реквизит(''+idAttr).Периодический)*1>0 then begin
          // Вызываем диалоговое окно просмотра истории пер. рекв.
                 Dlg_AttrPeriod_History.Execute(idRef,idAttr,sCode);
              end
              else begin
           ShowMessage('Этот реквизит:'+idAttr+'  не периодический');
              end;
           end
           else begin
              ShowMessage('Реквизит:'+idAttr+'  не существует');
           end;
        end
        else begin
          ShowMessage('Не выбран элемент справочника');
        end;
     end
     else begin
       ShowMessage('Не выбран элемент или реквизит справочника');
     end;
  end
```

```
  else begin
    ShowMessage('Ссылка на справочник — не корректна');
  end;
end;
```

Поясним текст обработчика. Здесь все прозрачно, пояснений требуют лишь две строки:

1. Проверка, является ли реквизит справочника периодическим

   ```
   if (oleBuf.Реквизит(''+idAttr).Периодический)*1>0 then begin
   ```

2. Вызов созданной нами формы для просмотра истории значений периодического реквизита (рис. 4.89).

   ```
   Dlg_AttrPeriod_History.Execute(idRef,idAttr,sCode);
   ```

Скомпилируем проект и запустим на выполнение. Сделаем выборку элементов справочника и поэкспериментируем с программой в рамках уже сделанных нами разработок. Для начала выделим ячейку в строке с кодом **5** и столбце **ДатаРождения** (рис. 4.91).

На экране появится сообщение (рис. 4.92).

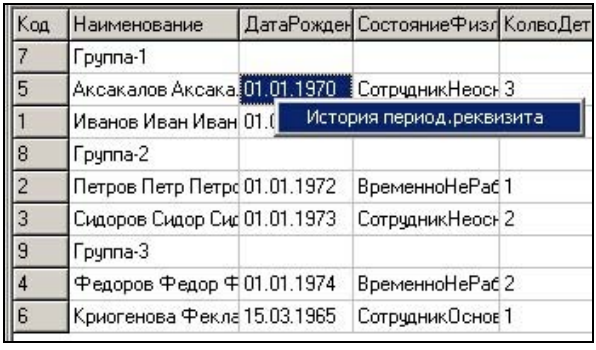



**Рис. 4.91.** Вызов меню для просмотра истории значений периодического реквизита справочника

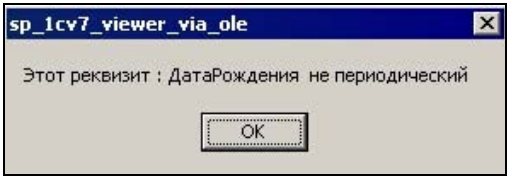



**Рис. 4.92.** Сообщение о некорректности обращения к реквизиту как к периодическому

В данном случае сработала проверка на корректность (листинг 4.22) в строке:

```
if (oleBuf.Реквизит(''+idAttr).Периодический)*1>0 then begin
```

То есть реквизит `ДатаРождения` действительно не периодический.

Теперь то же самое повторим для ячейки в строке с кодом **6** и столбце **КолвоДетей** (рис. 4.93).

**Рис. 4.93.** Вызов меню для просмотра истории значений периодического реквизита справочника

На экране появится разработанная выше форма просмотра истории значений периодического реквизита справочника (рис. 4.94). Тип значения: `Базовый` — `Число`.

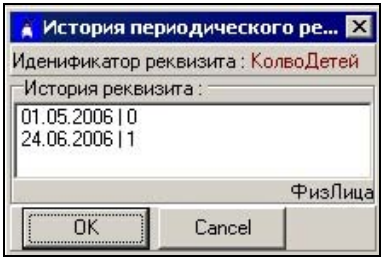



**Рис. 4.94.** История значений периодического реквизита `КолвоДетей`

Затем отметим ячейку в столбце `СостояниеФизлица` и вновь вызовем меню "История периодического реквизита" (рис. 4.95).

На экране появится форма, где отражена история значений периодического реквизита типа `Агрегатный` — `Перечисление` (рис. 4.96).

**Рис. 4.95.** Вызов меню для просмотра истории значений периодического реквизита справочника

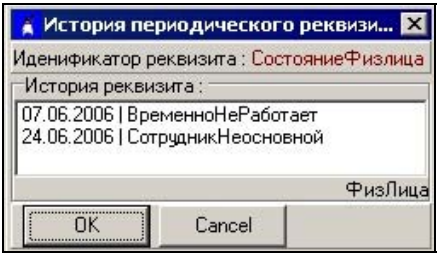



**Рис. 4.96.** История значений периодического реквизита `СостояниеФизлица`

В данном случае мы рассмотрели вариант, когда значение периодического реквизита изменяется вручную. Но часто для изменения значений периодических реквизитов справочников используют механизм проведения документов (более подробно о проведении документов можно прочитать в специальной литературе по системе 1С:Предприятие).

Чтобы продемонстрировать способ считывания истории значений периодических реквизитов справочников с учетом документов, вновь доработаем нашу конфигурацию. Для этого запустим Конфигуратор 1С:Предприятие и откроем нашу конфигурацию (см. рис. 2.8).

Создадим новый тип документов, которые при проведении будут выполнять следующие простые действия: для указанных физических лиц (элементов справочника `ФизЛица`) будут изменены значения периодического реквизита `СостояниеФизлица` на новые, указанные в документе.

В дереве конфигурации выберем ветку "Журналы документов" и создадим новый журнал документов `ЖурналДокументовФизЛиц` (рис. 4.97).

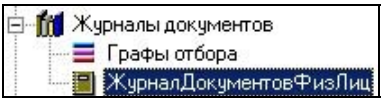



**Рис. 4.97.** Новый журнал документов в конфигурации 1С:Предприятие

Далее, в дереве конфигурации выберем ветку "Документы" и внесем новый реквизит в раздел "Общие реквизиты" (рис. 4.98).

На рис. 4.99 представлены свойства нового реквизита.

На рис. 4.100 показано дерево конфигурации после внесенных изменений.

Теперь создадим новый тип документов `ИзменениеСостоянияФизЛиц` (рис. 4.101 и 4.102).

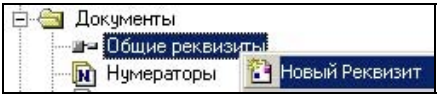



**Рис. 4.98.** Новый общий реквизит документов в конфигурации системы

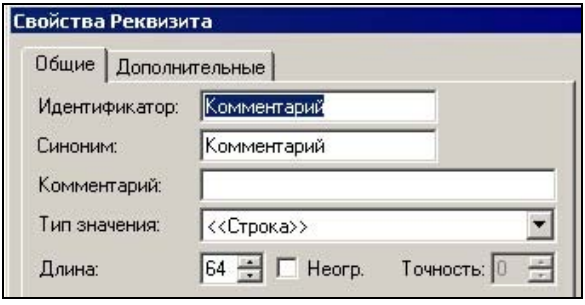



**Рис. 4.99.** Свойства общего реквизита документов `Комментарий`

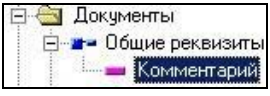



**Рис. 4.100.** Список общих реквизитов документов

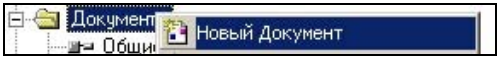



**Рис. 4.101.** Создание нового типа документов

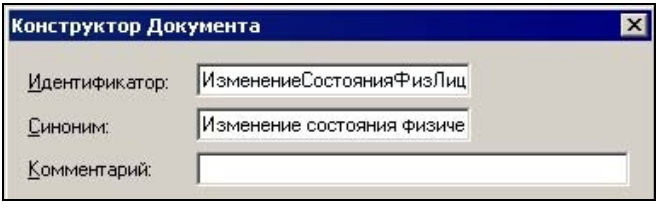



**Рис. 4.102.** Идентификатор нового типа документов

Укажем, что создаваемый нами тип документов должен поддерживать механизм проведения и при сохранении документа должен перепроводиться (рис. 4.103).

***Примечание***

Более подробную информацию о механизме проведения документов можно найти в специальной литературе по системе 1С:Предприятие.

На рис. 4.104 представлено диалоговое окно настройки свойств документов типа `ИзменениеСостоянияФизлиц`.

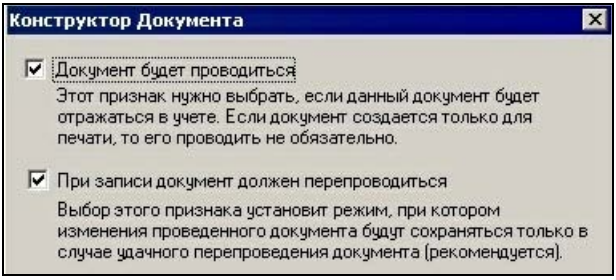



**Рис. 4.103.** Способность документов данного типа поддерживать механизм проведения

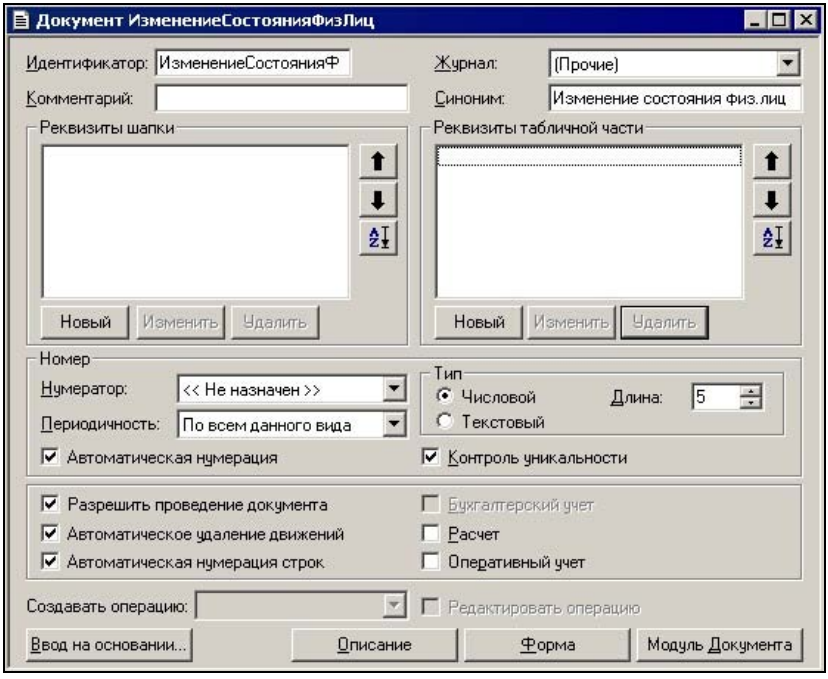



**Рис. 4.104.** Окно настройки свойств документов

***Примечание***

При создании у нового документа есть только два обязательных свойства, которые создаются системой 1С:Предприятие автоматически для любого типа документов, — дата и номер документа.

Документы состоят из двух частей: шапка документа и многострочная табличная часть, поэтому реквизиты документа также делятся на `Реквизиты шапки` и `Реквизиты табличной части`.

Внесем реквизиты табличной части документов этого типа (рис. 4.105 и 4.106).

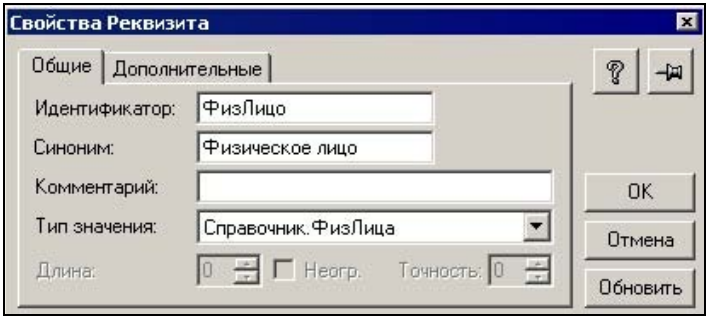



**Рис. 4.105.** Реквизит `ФизЛицо` табличной части документов типа `ИзменениеСостоянияФизЛиц`

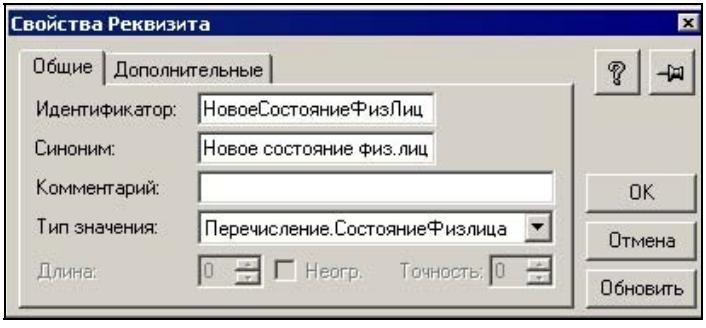



**Рис. 4.106.** Реквизит `НовоеСостояниеФизЛиц` табличной части документов типа `ИзменениеСостоянияФизЛиц`

***Примечание***

Реквизит документа `ФизЛицо` — это ссылка на элемент справочника `ФизЛица`, а реквизит документа `НовоеСостояниеФизЛиц` — это новое значение периодического реквизита `СостояниеФизлица` элемента справочника, на который ссылается реквизит документа `ФизЛицо`.

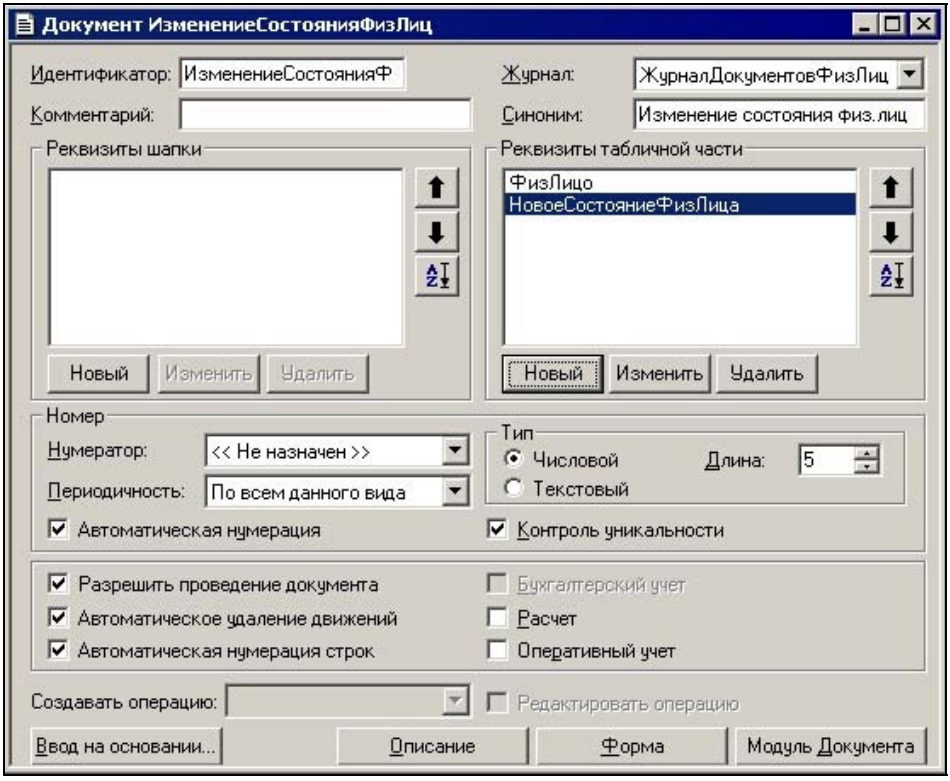



**Рис. 4.107.** Окно настройки свойств документов типа `ИзменениеСостоянияФизлиц`

Далее в раскрывающемся списке **Журнал** выберем созданный нами журнал **ЖурналДокументовФизЛиц** (рис. 4.107).

Теперь нужно написать текст процедуры, где будет реализован алгоритм проведения документов данного типа. Для этого нужно нажать кнопку **Модуль Документа** (см. рис. 4.107) и с помощью языка программирования системы 1С:Предприятие реализовать алгоритм проведения документов данного типа. В листинге 4.23 представлен текст процедуры, реализующей простейший алгоритм проведения документа.

**Листинг 4.23. Процедура обработки проведения документа ИзменениеСостоянияФизЛиц**

```
Процедура ОбработкаПроведения()
 ВыбратьСтроки();
 Пока ПолучитьСтроку() = 1 Цикл
```

```
    ПривязыватьСтроку(НомерСтроки);
    УстановитьРеквизитСправочника(
                                    ФизЛицо,
                                    "СостояниеФизлица",
                                    НовоеСостояниеФизЛица,
                                    ДатаДок
                                    );
 КонецЦикла;
КонецПроцедуры
```

В табл. 4.5 приведено краткое пояснение к каждой строке процедуры.

***Таблица 4.5.*** *Назначение строк процедуры обработки проведения документов типа* `ИзменениеСостоянияФизлиц`

| Строка процедуры | Назначение |
|---|---|
| `Процедура ОбработкаПроведения()` | Предопределенная процедура модуля документа |
| `ВыбратьСтроки;` | Открывает выборку строк табличной части данного документа |
| `Пока ПолучитьСтроку() = 1 Цикл` | Позиционирование на каждой строке документа (из выбранных) в цикле |
| `ПривязыватьСтроку(НомерСтроки);` | Означает, что при выполнении движений, создаваемых документом (например, по регистрам или при записи значений периодических реквизитов), движения будут записываться с привязкой не только к документу, но и к номеру строки табличной части документа. Это бывает крайне полезно, когда в документе много строк |
| `УстановитьРеквизитСправочника` | Записывается значение периодического реквизита справочника с привязкой к документу (и строке документа в данном случае) |

***Примечание***

Более подробно методы и свойства документов описаны в специальной литературе по системе 1С:Предприятие.

Теперь создадим форму документов типа `ИзменениеСостоянияФизлиц`. Для этого нужно нажать кнопку **Форма** (см. рис. 4.107).

В диалоговом окне настройки размещения реквизитов документа в форме (рис. 4.108) нужно отметить все реквизиты документа и нажать кнопку **Вставить**.

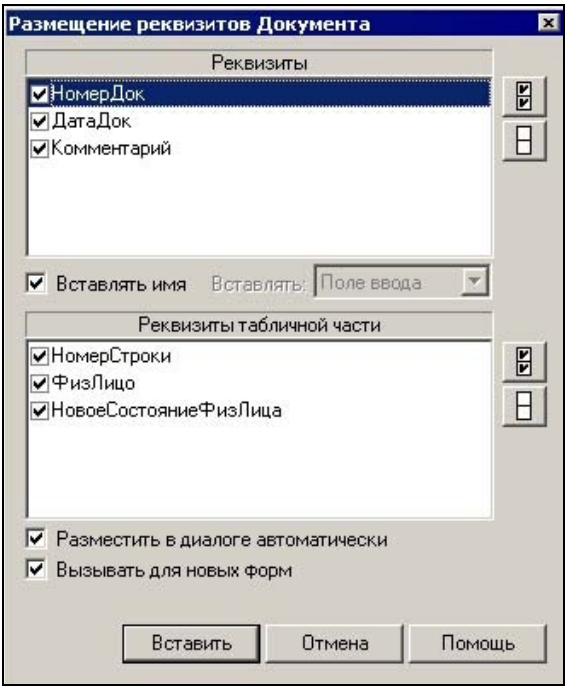



**Рис. 4.108.** Окно настройки размещения реквизитов документа в форме документа

После автоматического размещения соответствующих элементов управления в форме документа нужно самостоятельно их переместить на новую позицию или выровнять, если требуется (рис. 4.109).

Сохраним изменения в конфигурации (см. рис. 2.17) и запустим программу 1С:Предприятие (см. рис. 2.23). В меню программы 1С:Предприятие выберем пункт **Операции > Журналы документов** (рис. 4.110).

В открывшемся списке журналов выберем тот, к которому привязаны документы типа `ИзменениеСостоянияФизЛиц` (рис. 4.111).

Для того чтобы внести новый документ, вызовем всплывающее меню в списке документов журнала и выберем пункт **Новый** (рис. 4.112).

В открывшейся форме документа введем информацию, как указано на рис. 4.113, и нажмем кнопку **OK**.

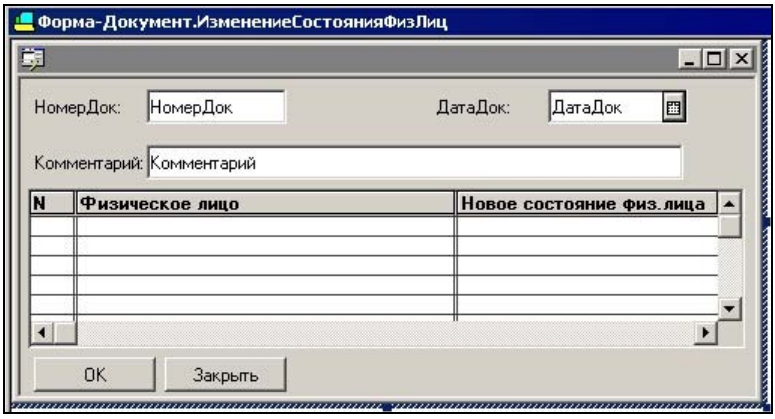



**Рис. 4.109.** Форма документов вида `ИзменениеСостояниеФизЛиц`

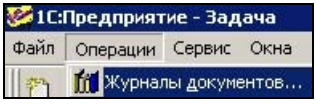



**Рис. 4.110.** Меню для выбора списка журналов документов

**Рис. 4.111.** Для того чтобы работать с документами, выберем соответствующий журнал

**Рис. 4.112.** Для того чтобы ввести новый документ, выберем в соответствующем журнале пункт **Новый** всплывающего меню

**Рис. 4.113.** Документ типа `ИзменениеСостояниеФизлиц` в режиме редактирования информации

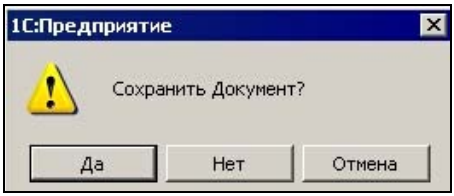



**Рис. 4.114.** Запрос на сохранение изменений в документе

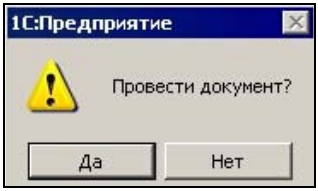



**Рис. 4.115.** Запрос на проведение документа

Подтвердим запрос на сохранение (рис. 4.114) и проведение (рис. 4.115) документа.

Новый документ появится в журнале документов. Чтобы посмотреть движения, созданные этим документом при проведении, нужно вызвать всплывающее меню и выбрать пункт **Движения документа** (рис. 4.116).

**Рис. 4.116.** Пункт меню для вызова диалогового окна просмотра движений, созданных данным документом

На экране появится диалоговое окно для выбора вида движений, созданных данным документом (рис. 4.117).

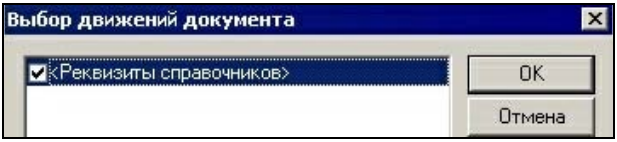



**Рис. 4.117.** Список видов движений, созданных данным документом

В данном случае, согласно принятому нами алгоритму проведения документов типа `ИзменениеСостоянияФизЛиц` (см. листинг 4.23), были созданы движения, связанные только с записью новых значений периодического реквизита `СостояниеФизлица` справочника `ФизЛица`. После выбора вида движения на экране появится таблица (рис. 4.118), где перечислены новые значения периодического реквизита `СостояниеФизлица`, установленные при проведении данного документа.

| | I | Справочник | Реквизит | Элемент | Дата | Значение |
|---|---|---|---|---|---|---|
| | | Список физических ли | Состояние физл | Аксакалов Аксакал А | 28.06.06 | Сотрудник по основному |
| | | Список физических ли | Состояние физл | Федоров Федор Фед | 28.06.06 | Сотрудник по неосновном |
| | | Список физических ли | Состояние физл | Криогенова Фекла М | 28.06.06 | Временно не работает |

**Рис. 4.118.** Значения периодического реквизита, установленные документом

Посмотрим, каким образом изменилась история значений периодического реквизита в справочнике после проведения документа. Для этого откроем справочник `ФизЛица`, выберем строку с кодом 5 (рис. 4.119).

| | К... | Наименование | Дата ... | Состояние физлица |
|---|---|---|---|---|
| | 5 | Аксакалов Аксакал Ак | 01.01.70 | Сотрудник по основному месту работы |
| | 7 | Группа-1 | | |
| | 8 | Группа-2 | | |
| | 9 | Группа-3 | | |
| | 1 | Иванов Иван Иванович | 01.01.71 | Сотрудник по основному месту работы |
| | 6 | Криогенова Фекла Ма | 15.03.65 | Временно не работает |
| | 2 | Петров Петр Петрович | 01.01.72 | Временно не работает |
| | 3 | Сидоров Сидор Сидоро | 01.01.73 | Сотрудник по неосновному месту работы |
| | 4 | Федоров Федор Федо | 01.01.74 | Сотрудник по неосновному месту работы |

**Рис. 4.119.** Чтобы посмотреть, как изменилась история значений реквизита после проведения документа, выберем строку с кодом 5 в справочнике `ФизЛица`

Элемент справочника `ФизЛица` с кодом 5 присутствует в табличной части проведенного документа (см. рис. 4.113). Следовательно, история значений периодического реквизита `СостояниеФизлица` (рис. 4.120) этого элемента справочника была изменена проведенным документом.

| К... | Наименование | Дата... | Состояние физлица |
|---|---|---|---|
| 5 | Аксакалов Аксакал А | 01.01.70 | Сотрудник по основному месту работы |
| 7 | Группа-1 | | |
| 8 | Группа-2 | | |
| 9 | Группа-3 | | |

Новый
История значения

**Рис. 4.120.** Выбор пункта меню для просмотра истории значений периодического реквизита справочника

В диалоговом окне (рис. 4.121) выберем периодический реквизит `СостояниеФизлица`.

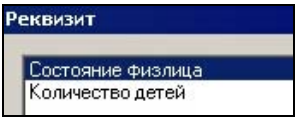
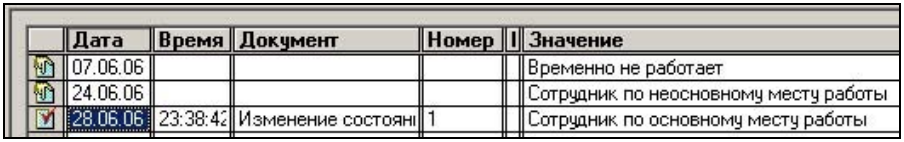



**Рис. 4.121.** Выбор периодического реквизита для просмотра истории значений

На экране появится окно истории изменения значений выбранного периодического реквизита (рис. 4.122).



| | Дата | Время | Документ | Номер | I | Значение |
|---|---|---|---|---|---|---|
| | 07.06.06 | | | | | Временно не работает |
| | 24.06.06 | | | | | Сотрудник по неосновному месту работы |
| | 28.06.06 | 23:38:42 | Изменение состояні | 1 | | Сотрудник по основному месту работы |

**Рис. 4.122.** История значений периодического реквизита `СостояниеФизлица` в справочнике `ФизЛица` для элемента с кодом 5

Как видите, в истории периодического реквизита есть значения, которые были изменены вручную, но последнее значение (см. рис. 4.122) было изменено документом, который был создан и проведен (см. рис. 4.112—4.115).

***Примечание***

Возможность изменения значений реквизита вручную и/или при проведении документов указывается при формировании конфигурации 1С:Предприятие. В данном случае при создании периодического реквизита `СостояниеФизлица` в справочнике `ФизЛица` были разрешены оба способа (см. рис. 4.109).

Вернемся к нашему проекту (Delphi). Посмотрим, как с помощью механизма OLE Automation можно прочитать информацию о том, каким именно способом изменялись значения периодических реквизитов. Для этого изменим метод `Execute` формы (см. рис. 4.89) следующим образом (листинг 4.24). Добавим две переменные:

- `ole_doc` — ссылка на объект типа `Документ`, при проведении которого было создано значение реквизита справочника `ФизЛица`;
- `sNameDoc` — строка, где будем формировать наименование этого документа.

**Листинг 4.24. Изменения в методе `Execute` формы "История периодического реквизита"**

```
function TDlg_AttrPeriod_History.Execute(idRef,idAttrPeriod,sCode:
string): boolean;
Var
  ole_doc : OleVariant; // Ссылка на "ТекущийДокумент"
  sNameDoc: string;     // Наименование документа
```

Дополним текст метода `Execute` (листинг 4.21) следующими строками (листинг 4.25).

**Листинг 4.25. Изменения в методе `Execute` формы "История периодического реквизита"**

```
  ...
  if NrType=11 then begin // Справочник
     sValAttr:= Trim(ole_period.Наименование);
  end;
//=========================================================
  sNameDoc:='';
  ole_doc:=ole_period.ТекущийДокумент;
  if length(Trim(ole_doc.Вид))>0 then begin
     sNameDoc:=Trim(ole_doc.Вид)
             + '  N '
             + Trim(ole_doc.НомерДок)
             + ' ['
             + Trim(ole_doc.ДатаДок)
             + '] '
             + Trim(ole_doc.Комментарий);
  end;
  if length(sNameDoc)>0 then begin
    ListBox1.Items.ADD(Trim(ole_period.ДатаЗнач)+' | '+sValAttr+' <-
      '+sNameDoc);
  end
  else begin
     ListBox1.Items.ADD(Trim(ole_period.ДатаЗнач)+' | '+sValAttr);
  end;
```

Поясним текст метода `Execute` (листинг 4.25).

Как видите, ссылку на документ, сформировавший значение периодического реквизита при проведении, можно получить, обратившись к свойству `ТекущийДокумент` объекта `Периодический` (см. выше в этом разделе).

Это выполняется в строке:

```
ole_doc:=ole_period.ТекущийДокумент;
```

Если ссылка на `ТекущийДокумент` актуальна, то метод `Вид` текущего документа возвращает непустую строку. В этом случае для формирования наименования документа можно обратиться к соответствующим свойствам текущего документа:

```
sNameDoc:=Trim(ole_doc.Вид)
            + '  N '
            + Trim(ole_doc.НомерДок)
            + ' ['
            + Trim(ole_doc.ДатаДок)
            + '] '
            + Trim(ole_doc.Комментарий);
```

Сохраним наши изменения, скомпилируем и запустим проект на выполнение, после чего произведем выборку элементов справочника `ФизЛица` (рис. 4.123).

| Код | Наименование | ДатаРожде | СостояниеФизлица |
|---|---|---|---|
| 7 | Группа-1 | | |
| 5 | Аксакалов Аксакал Ак | 01.01.1970 | СотрудникОсновной |
| 1 | Иванов Иван Иванович | 01.01.1971 | СотрудникОсновной |
| 8 | Группа-2 | | |
| 2 | Петров Петр Петрович | 01.01.1972 | ВременноНеРаботает |
| 3 | Сидоров Сидор Сидоро | 01.01.1973 | СотрудникНеосновной |
| 9 | Группа-3 | | |
| 4 | Федоров Федор Федо | 01.01.1974 | СотрудникНеосновной |
| 6 | Криогенова Фекла Ма | 15.03.1965 | ВременноНеРаботает |

**Рис. 4.123.** Выборка элементов справочника `ФизЛица`

Далее отметим ячейку в строке с кодом 5 и столбце **СостояниеФизлица** (рис. 4.124).

На экране появится разработанная нами Delphi-форма, где отражена история изменения значений периодического реквизита справочника с учетом проведенных документов (рис. 4.125).

**Рис. 4.124.** Вызов формы для просмотра истории значений периодического реквизита `СостояниеФизлица` элемента справочника с кодом 5

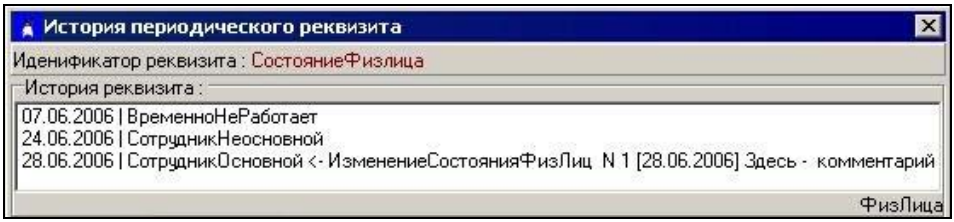



**Рис. 4.125.** История изменений периодического реквизита `СостояниеФизлица`

Результат, представленный на рис. 4.125, можно сравнить с тем, что показано на рис. 4.122. Как видите, информация, считанная с использованием механизма OLE Automation, совпадает с той, что представлена путем визуализации истории изменения значений периодического реквизита стандартными средствами системы 1С:Предприятие.

Созданные нами изменения в конфигурации отражены в словаре данных (файл 1Cv7.DD) следующим образом (рис. 4.126).

```
#
#==============================================================================
#==TABLE no 11     : Документ ИзменениеСостоянияФизЛиц
# Name     |Descr                                |Type[A/S/U]|DBTableName|ReUsable
T=DH46     |Документ ИзменениеСостоянияФиз|A          |DH46       |1
#-----Fields-------
# Name       |Descr                |Type|Length|Precision
F=IDDOC      |ID Document's        |C   |9     |0
F=SP49       |(P)Комментарий       |C   |64    |0
#----Indexes------
# Name      |Descr          |Unique|Indexed fields                                                    |DBName
I=ID        |of IDDOC       |0     |IDDOC                                                             |ID
#
#==============================================================================
#==TABLE no 12     : Документ (Мн.ч.) ИзменениеСостоянияФизЛиц
# Name     |Descr                                |Type[A/S/U]|DBTableName|ReUsable
T=DT46     |Документ (Мн.ч.) ИзменениеСост|A          |DT46       |1
#-----Fields-------
# Name       |Descr                |Type|Length|Precision
F=IDDOC      |ID Document's        |C   |9     |0
F=LINENO     |LineNo               |N   |4     |0
F=SP50       |(P)ФизЛицо           |C   |9     |0
F=SP51       |(P)НовоеСостояниеФиз|C   |9     |0
#----Indexes------
# Name      |Descr          |Unique|Indexed fields                                                    |DBName
I=IDLINE    |of IDDOC+LineN|0     |IDDOC,LINENO                                                      |IDLINE
```

**Рис. 4.126.** Содержание файла 1Cv7.DD.
Структура документа `ИзменениеСостоянияФизЛиц`

Как видите, для каждого нового типа документов создаются две таблицы. Одна таблица (с префиксом `DH`) соответствует "шапке" документа, другая (с префиксом `DT`) — "табличной части" документа. Более подробно об этом — в *части II*.

С помощью любого просмотрщика DBF-файлов откроем журнал документов (файл 1SJOURN.DBF, рис. 4.127).

Содержание таблицы :
1SJOURN
Открыть таблицу

| IDJOURNAL | IDDOC | IDDOCDEF | APPCODE | DATE | TIME | DNPREFIX | DOCNO | CLOSED | ISMARK | ACTCNT | VERSTAMP |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 18 | 2 | 1A | 8 | 28.06.2006 | E2SLK0 | 46 | 1 | 1 | | 3 | 2 |

**Рис. 4.127.** Журнал документов (файл 1SJOURN.DBF)

В журнале всего одна строка, соответствующая созданному документу. Здесь следует отметить, что поле `IDDOC` содержит значение 2. Это связано с тем, что в процессе работы над книгой был создан еще один документ, но затем удален.

Откроем файл 1SCONST.DBF (где система 1С:Предприятие размещает значения периодических реквизитов). Как видите, строки, созданные при проведении документов, имеют соответствующие ссылки на `ТекущийДокумент` в поле `DOCID` (рис. 4.128).

Содержание таблицы :
1SCONST
Открыть таблицу

| OBJID | ID | DATE | PARTNO | VALUE | DOCID | TIME | ACTNO | LINENO | TVALUE |
|---|---|---|---|---|---|---|---|---|---|
| 5 | 16 | 24.06.2006 | 0 | 14 | 0 | 0 | 0 | 0 | |
| 1 | 16 | 24.06.2006 | 0 | 13 | 0 | 0 | 0 | 0 | |
| 4 | 16 | 24.06.2006 | 0 | 15 | 0 | 0 | 0 | 0 | |
| 6 | 16 | 24.06.2006 | 0 | 13 | 0 | 0 | 0 | 0 | |
| 2 | 16 | 24.06.2006 | 0 | 15 | 0 | 0 | 0 | 0 | |
| 6 | 17 | 24.06.2006 | 0 | 1 | 0 | 0 | 0 | 0 | |
| 3 | 17 | 24.06.2006 | 0 | 2 | 0 | 0 | 0 | 0 | |
| 5 | 16 | 28.06.2006 | 0 | 13 | 2 | E2SLK0 | 1 | 0 | |
| 4 | 16 | 28.06.2006 | 0 | 14 | 2 | E2SLK0 | 2 | 0 | |
| 6 | 16 | 28.06.2006 | 0 | 15 | 2 | E2SLK0 | 3 | 0 | |

**Рис. 4.128.** Значения периодических реквизитов справочников размещаются в файле 1SCONST.DBF

Откроем файл DH46.DBF ("шапка" документа `ИзменениеСостоянияФизЛиц`). На рис. 4.129 представлено содержание "шапки" документа.

Содержание таблицы :
DH46
Открыть таблицу

| IDDOC | SP49 |
|---|---|
| 2 | Здесь - комментарий |

**Рис. 4.129.** "Шапка" документа типа `ИзменениеСостоянияФизЛиц` (файл DH46.DBF)

Как видите, в этой таблице хранятся только системный идентификатор строки (поле IDDOC), соответствующей конкретному документу, и реквизиты "шапки" документов. Свойства `Дата` и `Время` этого документа хранятся в журнале документа (см. рис. 4.127).

Теперь откроем табличную часть документов этого типа (файл DT46.DBF). Как видите (рис. 4.130), в этой таблице хранятся системный идентификатор строки (поле IDDOC), соответствующей конкретному документу, номер строки документа (поле LINENO) и реквизиты "табличной части" документов.

Содержание таблицы :
DT46
Открыть таблицу

| IDDOC | LINENO | SP50 | SP51 |
|---|---|---|---|
| 2 | 1 | 5 | 13 |
| 2 | 2 | 4 | 14 |
| 2 | 3 | 6 | 15 |

**Рис. 4.130.** "Табличная часть" документа, типа `ИзменениеСостоянияФизЛиц` (файл DT46.DBF)

Эта информация пригодится нам в *части II*, когда мы будем рассматривать возможность доступа к базе данных системы 1С:Предприятие с использованием SQL-запросов. В этом разделе не были затронуты вопросы визуализации реквизитов неопределенного типа. Их мы коснемся в *разд. 4.7.1*.

***Примечание***

Напомню, что представленные примеры исходных текстов — это упрощенная иллюстрация к рассматриваемым вопросам. Назначение этих примеров — дать представление о возможном подходе при решении задач информационного обмена с системой 1С:Предприятие. При работе с реальными базами 1С может потребоваться их дальнейшая доработка.

# 4.2. Считывание информации из документов

Документ — важное понятие системы 1С:Предприятие (подробную информацию о документах можно найти в специальной литературе по системе 1С:Предприятие). Нас в этом разделе будут интересовать только способы считывания информации о документах с использованием механизма OLE Automation.

Откроем Конфигуратор 1С:Предприятие и внесем следующие изменения в нашу конфигурацию:

1. Создадим новое перечисление `ПриоритетДокумента` (рис. 4.131).
2. Внесем новый реквизит типа `Перечисление.ПриоритетДокумента` в шапку документа `ИзменениеСостоянияФизЛиц` (рис. 4.132).

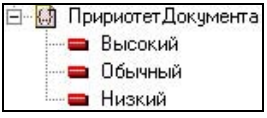



**Рис. 4.131.** Перечисление, определяющее приоритет документа

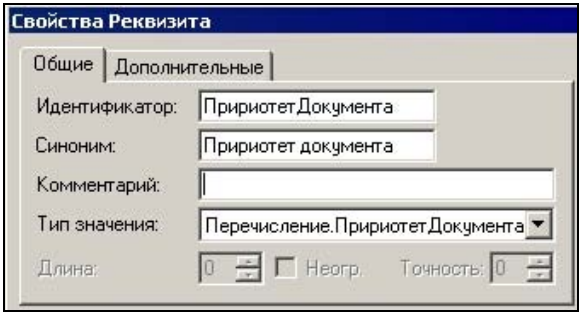



**Рис. 4.132.** Новый реквизит шапки документа типа `Перечисление.ПриоритетДокумента`

На рис. 4.133 представлена структура документа `ИзменениеСостоянияФизЛиц` после внесенных изменений.

Сохраним изменения в конфигурации (см. рис. 2.17).

Попробуем прочитать информацию о структуре документа (см. рис. 4.133), используя механизм OLE Automation. Для этого создадим новый проект (Delphi).

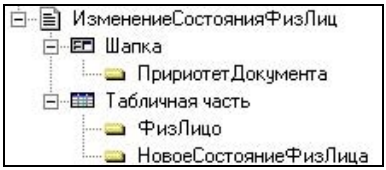



**Рис. 4.133.** Список реквизитов документа `ИзменениеСостояниеФизЛиц`

***Примечание***

Пример есть в папке \Part_1\Chapter_04\Topic_4_2\sp_LoadAttrFromDoc.PAS на сопроводительном компакт-диске книги.

На рис. 4.134 представлена форма проекта.

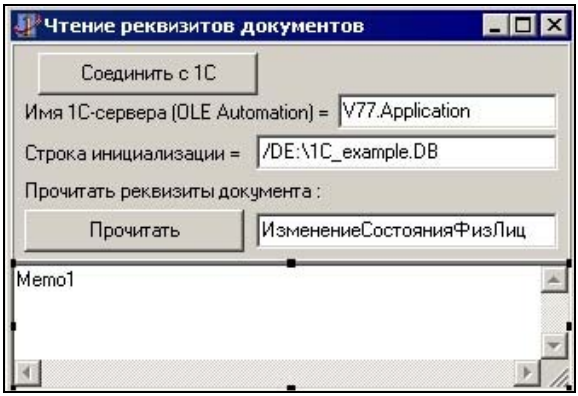



**Рис. 4.134.** Форма главного окна приложения sp_LoadAttrFromDoc

Этот проект создан по подобию проектов, рассмотренных ранее, поэтому пояснения не требуются. Интерес представляет только обработчик события "нажатие кнопки **Прочитать**". Но прежде чем написать текст обработчика, попробуем разобраться в вопросе, какие именно свойства и методы документов, необходимые для решения этой задачи, доступны через механизм OLE Automation. Мы уже знаем, что все свойства (реквизиты) документов можно разделить на три категории:

- общие реквизиты документов;
- реквизиты "шапки" документов;
- реквизиты "табличной части" документов.

Напишем функции, которые позволят прочитать список идентификаторов реквизитов документов каждой из этих трех категорий.

В листинге 4.26 представлен текст функции, считывающей список идентификаторов общих реквизитов всех документов.

**Листинг 4.26. Получение списка общих реквизитов документов**

```
function Get_DocAttrCommon_to_List(handle1C:OleVariant; List:TStrings):
integer;
// Получить список значений общих реквизитов документов
// handle1C — ссылка на 1С:Предприятие
// List     — список, в котором возвращаются общие реквизиты документов
Var
  Yes: byte;  // Вспомогательная переменная
  oleBuf: OleVariant; // Вспомогательная переменная
  sValAttr,   // Значение реквизита (строка)
  KindAttr,   // Вид реквизита
  TypeAttr,   // Тип реквизита
  idAttr: string; // Идентификатор реквизита
  iAttr,
  CountAttr: integer; // Количество общих реквизитов документов
begin
  Result:=0;
  Yes:= 0;
  if List<>NIL then begin
     List.Clear;
     if not VarIsEmpty(handle1C) then begin
        Yes:=1;
     end;
  end;
  if Yes>0 then begin
     // Количество общих реквизитов документов
     CountAttr:=1*handle1C.Метаданные.ОбщийРеквизитДокумента;
     if CountAttr>0 then begin
        iAttr:=0;
        while iAttr<CountAttr do
         begin
          iAttr:=iAttr+1;
          // Читаем ссылку на реквизит табличной части
          oleBuf:=handle1C.Метаданные.ОбщийРеквизитДокумента(1*iAttr);
          // Считываем идентификатор реквизита
          idAttr:=Trim(oleBuf.Идентификатор);
```

```
          // Считываем вид реквизита
          KindAttr:=Trim(oleBuf.Вид);
          // Считываем тип реквизита
          TypeAttr:=Trim(oleBuf.Тип);
          List.Values[idAttr]:='Вид:'+KindAttr+';  Тип:'+TypeAttr;
        end;
        Result:=List.Count;
      end;
  end;
end;
```

Поясним текст функции `Get_DocAttrCommon_to_List`. Как видите, и здесь основную роль играет объект `Метаданные`.

В функцию `Get_DocAttrCommon_to_List` передаются два параметра.

- `handle1C` — ссылка на 1С:Предприятие (сервер OLE Automation);
- `List` — список, в котором возвращаются общие реквизиты документов.

Текст функции достаточно прост, поэтому отметим лишь основные моменты.

- Если метод `ОбщийРеквизитДокумента`:

  `handle1C.Метаданные.`**`ОбщийРеквизитДокумента`**`;`

  вызывается без параметров, то возвращается количество общих реквизитов документов.

- Если метод `ОбщийРеквизитДокумента`:

  `oleBuf:=handle1C.Метаданные.`**`ОбщийРеквизитДокумента`**`(1*`**`iAttr`**`);`

  вызывается с параметром (номер общего реквизита документов), то возвращается ссылка на общий реквизит документов, соответствующий указанному номеру.

- Получив ссылку на общий реквизит документов как объект, можно обращаться к его свойствам и методам.

В данном случае мы используем свойства реквизита: `Идентификатор`, `Вид`, `Тип`.

Теперь напишем функцию, которая позволит считывать список идентификаторов реквизитов "шапки" документов (листинг 4.27).

**Листинг 4.27. Получение списка реквизитов "шапки" документа**

```
function Get_DocAttrHead_to_
List(handle1C:OleVariant;idDoc:string;List:TStrings):integer;
// Получить список значений реквизитов "шапки" документа
```

```
// handle1C — ссылка на 1С:Предприятие
// idDoc    — идентификатор тестируемого документа
// List     — список, в котором возвращаются реквизиты "шапки" документа
Var
  Yes: byte;  // Вспомогательная переменная
  oleBuf: OleVariant; // Вспомогательная переменная
  sValAttr, // Значение реквизита (строка)
  KindAttr, // Вид реквизита
  TypeAttr, // Тип реквизита
  idAttr: string; // Идентификатор реквизита
  iAttr,
  CountAttr: integer; // Количество реквизитов "шапки" документов
begin
  Result:=0;
  Yes:= 0;
  if List<>NIL then begin
     List.Clear;
     if not VarIsEmpty(handle1C) then begin
        idDoc:=Trim(idDoc);
        if length(idDoc)>0 then begin
           // Проверяем существование документа с идентификатором idDoc
           if handle1C.Метаданные.Документ(''+idDoc).Выбран>0 then begin
              Yes:=1;
           end;
        end;
     end;
  end;
  if Yes>0 then begin
     // Количество реквизитов "шапки" документа
     CountAttr:=1*handle1C.Метаданные.Документ(''+idDoc).РеквизитШапки;
     if CountAttr>0 then begin
        iAttr:=0;
        while iAttr<CountAttr do
         begin
          iAttr:=iAttr+1;
          // Читаем ссылку на реквизит табличной части
oleBuf:=handle1C.Метаданные.Документ(''+idDoc).РеквизитШапки(1*iAttr);
          // Считываем идентификатор реквизита
          idAttr:=Trim(oleBuf.Идентификатор);
```

```
          // Считываем вид реквизита
          KindAttr:=Trim(oleBuf.Вид);
          // Считываем тип реквизита
          TypeAttr:=Trim(oleBuf.Тип);
          List.Values[idAttr]:='Вид:'+KindAttr+';  Тип:'+TypeAttr;
        end;
        Result:=List.Count;
      end;
  end;
end;
```

Поясним текст функции `Get_DocAttrHead_to_List`.

В функцию `Get_DocAttrHead_to_List` передаются три параметра:

- `handle1C` — ссылка на 1С:Предприятие (сервер OLE Automation);
- `idDoc` — идентификатор тестируемого документа;
- `List` — список, в котором возвращаются реквизиты шапки тестируемого документа.

Текст функции достаточно прост, поэтому отметим лишь основные моменты.

- Проверяется существование документа с идентификатором `idDoc`:

  ```
  if handle1C.Метаданные.Документ(''+idDoc).Выбран>0 then begin
  ```

- Если метод `РеквизитШапки`:

  ```
  handle1C.Метаданные.Документ(''+idDoc).РеквизитШапки;
  ```

  вызывается без параметров, то возвращается количество реквизитов шапки документа.

- Если метод `РеквизитШапки`:

  ```
  oleBuf:=handle1C.Метаданные.Документ(''+idDoc).РеквизитШапки(1*iAttr);
  ```

  вызывается с параметром (номер реквизита документа), то возвращается ссылка на реквизит шапки документа, соответствующий указанному номеру.

- Получив ссылку на реквизит документа как объект, можно обращаться к его свойствам и методам.

В данном случае мы используем свойства реквизита: `Идентификатор`, `Вид`, `Тип`.

Теперь напишем функцию, которая позволит считывать список идентификаторов реквизитов табличной части документов (листинг 4.28).

**Листинг 4.28. Получение списка реквизитов табличной части документа**

```
function
Get_DocAttrTable_to_List(handle1C:OleVariant;idDoc:string;List:TStrings):
integer;
// Получить список значений реквизитов табличной части документа
// handle1C — ссылка на 1С:Предприятие
// idDoc    — идентификатор тестируемого документа
// List     — список, в котором возвращаются реквизиты
// табличной части тестируемого документа
Var
  Yes: byte;  // Вспомогательная переменная
  oleBuf: OleVariant; // Вспомогательная переменная
  sValAttr, // Значение реквизита (строка)
  KindAttr, // Вид реквизита
  TypeAttr, // Тип реквизита
  idAttr: string; // Идентификатор реквизита
  iAttr,
  CountAttr: integer; // Количество реквизитов табличной части документов
begin
  Result:=0;
  Yes:= 0;
  if List<>NIL then begin
     List.Clear;
     if not VarIsEmpty(handle1C) then begin
        idDoc:=Trim(idDoc);
        if length(idDoc)>0 then begin
           // Проверяем существование документа с идентификатором idDoc
           if handle1C.Метаданные.Документ(''+idDoc).Выбран>0 then begin
              Yes:=1;
           end;
        end;
     end;
  end;
  if Yes>0 then begin
     // Количество реквизитов табличной части документа
CountAttr:=1*handle1C.Метаданные.Документ(''+idDoc).РеквизитТабличнойЧасти;
     if CountAttr>0 then begin
        iAttr:=0;
```

```
        while iAttr<CountAttr do
         begin
          iAttr:=iAttr+1;
          // Читаем ссылку на реквизит табличной части
oleBuf:=handle1C.Метаданные.Документ(''+idDoc).РеквизитТабличнойЧасти
   (1*iAttr);
          // Считываем идентификатор реквизита
          idAttr:=Trim(oleBuf.Идентификатор);
          // Считываем вид реквизита
          KindAttr:=Trim(oleBuf.Вид);
          // Считываем тип реквизита
          TypeAttr:=Trim(oleBuf.Тип);
          List.Values[idAttr]:='Вид:'+KindAttr+';  Тип:'+TypeAttr;
        end;
        Result:=List.Count;
     end;
     Result:=List.Count;
  end;
end;
```

Поясним текст функции `Get_DocAttrTable_to_List`. Она отличается от функции `Get_DocAttrHead_to_List` лишь тем, что вместо метода `РеквизитШапки` вызывается метод `РеквизитТабличнойЧасти`. В остальном тексты обеих функций идентичны.

> ***Примечание***
>
> В методы `РеквизитШапки`, `РеквизитТабличнойЧасти` и `ОбщийРеквизитДокумента` можно передавать не только переменные типа `Integer`, но и переменные типа `String`. Если передается переменная типа `Integer`, то это интерпретируется как порядковый номер реквизита. Если передается переменная типа `String`, то это интерпретируется как идентификатор реквизита. Далее мы будем использовать эти возможности.

Теперь, наконец, напишем текст обработчика события "нажатие кнопки **Прочитать**" (листинг 4.29).

**Листинг 4.29. Обработчик события "нажатие кнопки *Прочитать*"**

```
procedure TForm1.Button2Click(Sender: TObject);
// Прочитать список реквизитов документа:
// Edit3.Text — идентификатор тестируемого документа,
```

```
// ListBox1.Items — результат считывания реквизитов,
// тестируемого документа,
// List — вспомогательная переменная
Var
  List: TStrings;
begin
  ListBox1.Items.Clear;
  if not VarIsEmpty(Ole1C) then begin
     List:=TStringList.Create;
     TRY
       // Читаем список общих реквизитов документов
       ListBox1.Items.Add('Общие реквзиты:');
       Get_DocAttrCommon_to_List(Ole1C, List);
       ListBox1.Items.AddStrings(List);
       ListBox1.Items.Add('-------');
       // Читаем список реквизитов "шапки" документа
       ListBox1.Items.Add('Реквизиты шапки:');
       Get_DocAttrHead_to_List(Ole1C,Trim(Edit3.Text),List);
       ListBox1.Items.AddStrings(List);
       ListBox1.Items.Add('-------');
       // Читаем список реквизитов табличной части документа
       ListBox1.Items.Add('Реквизиты табличной части:');
       Get_DocAttrTable_to_List(Ole1C,Trim(Edit3.Text),List);
       ListBox1.Items.AddStrings(List);
     FINALLY
       if List<>NIL then begin
          List.Clear;
          List.Free;
          List:=NIL;
       end;
     END;
  end;
end;
```

Поясним текст обработчика.

Здесь поочередно вызываются созданные ранее функции (`Get_DocAttrCommon_to_List`, `Get_DocAttrHead_to_List`, `Get_DocAttrTable_to_List`) и результат считывания реквизитов документа записывается в `ListBox1.Items`.

Скомпилируем наш проект и запустим на выполнение. После соединения с системой 1С:Предприятие нажмем кнопку **Прочитать**. Результат представлен на рис. 4.135, пояснения излишни.

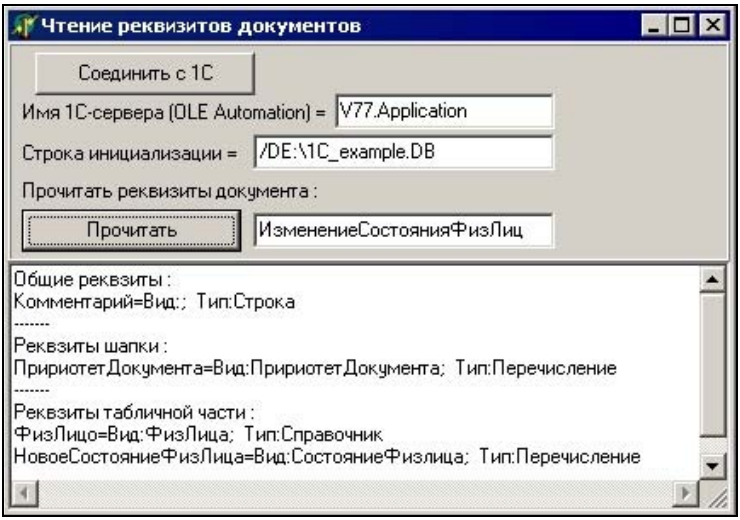



**Рис. 4.135.** Список реквизитов документа вида `ИзменениеСостоянияФизЛиц`

При обработке документов часто требуется проверить наличие реквизита (по идентификатору) в документах заданного вида. В глобальном модуле системы 1С:Предприятие для этого есть соответствующие "штатные" функции (листинги 4.30 и 4.33). Мы разработаем собственные функции, которые позволят выполнять эти же действия, но с помощью механизма OLE Automation.

Вот как в глобальном модуле 1С:Предприятие реализована проверка существования реквизита среди общих реквизитов документов и в "шапке" документа (листинг 4.30).

**Листинг 4.30. Проверка существования реквизита среди общих реквизитов и в "шапке" документа**

```
Функция глЕстьРеквизитШапки(ИмяРеквизита, ВидДокумента) Экспорт
Если ПустоеЗначение(ИмяРеквизита) = 1 Тогда
   Возврат 0;
ИначеЕсли Метаданные.ОбщийРеквизитДокумента(ИмяРеквизита).Выбран() = 1
Тогда
   Возврат 1;
ИначеЕсли ПустоеЗначение(ВидДокумента) = 1 Тогда
```

```
   Возврат 0;
ИначеЕсли Метаданные.Документ(ВидДокумента).РеквизитШапки(ИмяРеквизита).
Выбран()=1 Тогда
   Возврат 1;
Иначе
   Возврат 0;
КонецЕсли;
КонецФункции // глЕстьРеквизитШапки()
```

Здесь: `ИмяРеквизита` — идентификатор тестируемого реквизита; `ВидДокумента` — идентификатор тестируемого документа.

Текст функции прост и не требует пояснений. Единственное, что можно отметить, — в одной функции совмещены две проверки:

- проверка на существование тестируемого реквизита среди общих реквизитов документов;
- проверка на существование тестируемого реквизита среди реквизитов "шапки" документа.

При разработке своей (аналогичной) функции мы разделим эти проверки. В листинге 4.31 представлен текст функции, проверяющей существование тестируемого реквизита среди общих реквизитов документов.

**Листинг 4.31. Проверка существования реквизита среди общих реквизитов документов**

```
function YesAttrInDocCommon_1C(handle1C: OleVariant;idAttr: string): boolean;
// Проверка существования заданного реквизита среди общих
// реквизитов документов:
// handle1C — ссылка на 1С,
// idAttr — идентификатор тестируемого реквизита.
Var
  oleBuf: OleVariant;
begin
 Result:=FALSE;
 if not VarIsEmpty(handle1C) then begin
    idAttr:=Trim(idAttr);
    if length(idAttr)>0 then begin
        oleBuf:=handle1C.Метаданные.ОбщийРеквизитДокумента(''+idAttr);
        if oleBuf.Выбран>0 then begin
```

```
            Result:=TRUE;
         end;
        end;
    end;
 end;
end;
```

Текст этой функции не требует пояснений.

В листинге 4.32 представлен текст функции, проверяющей существование тестируемого реквизита среди реквизитов "шапки" документа заданного вида.

**Листинг 4.32. Проверка существования реквизита в "шапке" документа**

```
function YesAttrInDocHead_1C(handle1C: OleVariant;idDoc,idAttr: string):
boolean;
// Проверка существования заданного реквизита в "шапке" документа:
// handle1C — ссылка на 1С,
// idDoc — идентификатор тестируемого документа,
// idAttr — идентификатор тестируемого реквизита.
Var
  oleBuf: OleVariant;
begin
 Result:=FALSE;
 if not VarIsEmpty(handle1C) then begin
    idAttr:=Trim(idAttr);
    idDoc:=Trim(idDoc);
    if (length(idAttr)>0) and (length(idDoc)>0) then begin
oleBuf:=handle1C.Метаданные.Документ(''+idDoc).РеквизитШапки(''+idAttr);
       if oleBuf.Выбран >0 then begin
          Result:=TRUE;
       end;
    end;
 end;
end;
```

Текст этой функции также не требует пояснений.

Вот как в глобальном модуле 1С:Предприятие реализована проверка существования реквизита среди реквизитов табличной части документа (листинг 4.33).

**Листинг 4.33. Проверка существования реквизита в табличной части документа**

```
Функция глЕстьРеквизитМнЧ(ИмяРеквизита,ВидДокумента) Экспорт
Перем Реквизит;
Если (ПустоеЗначение(ВидДокумента) = 1) ИЛИ
     (ПустоеЗначение(ИмяРеквизита) = 1) Тогда
        Возврат 0;
КонецЕсли;
Реквизит=Метаданные.Документ(ВидДокумента).РеквизитТабличнойЧасти
(ИмяРеквизита);
Возврат Реквизит.Выбран();
КонецФункции // глЕстьРеквизитМнЧ()
```

В листинге 4.34 представлен текст нашей функции, выполняющей эти же действия.

**Листинг 4.34. Проверка существования реквизита в табличной части документа**

```
function YesAttrInDocTable_1C(handle1C: OleVariant;idDoc,idAttr: string):
boolean;
// Проверка существования заданного реквизита в табличной
// (многострочной) части документа:
// handle1C — ссылка на 1С,
// idDoc — идентификатор тестируемого документа,
// idAttr — идентификатор тестируемого реквизита.
Var
  oleBuf: OleVariant;
begin
 Result:=FALSE;
 if not VarIsEmpty(handle1C) then begin
    idAttr:=Trim(idAttr);
    idDoc:=Trim(idDoc);
    if (length(idAttr)>0) and (length(idDoc)>0) then begin
oleBuf:=handle1C.Метаданные.Документ(''+idDoc).РеквизитТабличнойЧасти(''+idAttr);
       if oleBuf.Выбран >0 then begin
          Result:=TRUE;
       end;
    end;
 end;
end;
```

Текст этой функции также не требует пояснений.

Теперь объединим все три наших функции в одну, которая позволит тестировать реквизит в целом по документу (листинг 4.35).

**Листинг 4.35. Проверка существования реквизита в документе**

```
function YesAttrInDoc_1C(handle1C: OleVariant;idDoc,idAttr: string):
integer;
// Проверка существования заданного реквизита в документе.
// Входные параметры:
// handle1C — ссылка на 1С,
// idDoc  — идентификатор тестируемого документа,
// idAttr — идентификатор тестируемого реквизита.
// Возвращаемое значение:
// 0 — нет такого реквизита,
// 1 — присутствует среди общих реквизитов документов,
// 2 — присутствует среди реквизитов "шапки" документа,
// 3 — присутствует среди реквизитов табличной части документа.
begin
  Result:=0;
  if YesAttrInDocCommon_1C(handle1C,idAttr) then begin
     Result:=1;
  end
  else begin
     if YesAttrInDocHead_1C(handle1C,idDoc,idAttr) then begin
        Result:=2;
     end
     else begin
        if YesAttrInDocTable_1C(handle1C,idDoc,idAttr) then begin
           Result:=3;
        end;
     end;
  end;
end;
```

Функция проста и не требует пояснений.

Проверим работоспособность наших функций. Для этого создадим обработчик события "двойной щелчок мышки на списке `ListBox1`" (листинг 4.36).

**Листинг 4.36. Проверка работоспособности функции `YesAttrInDoc_1C`**

```
procedure TForm1.ListBox1DblClick(Sender: TObject);
// Проверка работоспособности функции YesAttrInDoc_1C:
// Edit3.Text — идентификатор тестируемого документа,
// idAttr — идентификатор тестируемого реквизита.
Var
 idAttr: string; // Идентификатор тестируемого реквизита
 Sx: string;
 NrType: integer;
begin
  if ListBox1.ItemIndex>=0 then begin
    idAttr:=Trim(ListBox1.Items.NAMES[ListBox1.ItemIndex]);
    if InputQuery('Проверка реквизита','Идентификатор',idAttr) then begin
        idAttr:=Trim(idAttr);
        if length(idAttr)>0 then begin
           NrType:=YesAttrInDoc_1C(Ole1C,Edit3.Text,idAttr);
           if NrType=1 then begin
              Sx:= 'Реквизит <'
                  +idAttr
                  +'> относится к общим реквизитам документов';
              ShowMessage(Sx);
           end
           else begin
              if NrType=2 then begin
                 Sx:= 'Реквизит <'
                     +idAttr
                     +'> принадлежит шапке документа типа <'
                     +Edit3.Text+'>';
                 ShowMessage(Sx);
              end
              else begin
                 if NrType=3 then begin
                    Sx:= 'Реквизит <'
                        +idAttr
                        +'> принадлежит табличной части документа типа <'
                        +Edit3.Text+'>'
                    ShowMessage(Sx);
                 end
                 else begin
```

```
                    Sx:= 'Реквизит <'
                        +idAttr
                        +'> не корректен для документа типа <'
                        +Edit3.Text+'>'
                    ShowMessage(Sx);
                  end;
               end;
            end;
         end;
      end;
   end;
end;
```

Поясним текст обработчика. Здесь все просто.

1. Из текущей строки списка реквизитов (`ListBox1.Items`) считывается идентификатор реквизита:
   `idAttr:=Trim(ListBox1.Items.NAMES[ListBox1.ItemIndex]);`
2. Следует запрос на изменение тестируемого идентификатора реквизита:
   `if InputQuery('Проверка реквизита','Идентификатор',idAttr) then begin`
3. Вызывается функция `YesAttrInDoc_1C`:
   `NrType:=YesAttrInDoc_1C(Ole1C,Edit3.Text,idAttr);`
4. В зависимости от значения переменной `NrType` выводится соответствующее сообщение.

Скомпилируем и запустим на выполнение наш проект.

После того как будет получен список реквизитов документа `ИзменениеСостоянияФизЛиц` (см. ранее), сделаем двойной щелчок на строке, содержащей реквизит `Комментарий`. На экране появится запрос на изменение идентификатора реквизита. Удалим все лишнее кроме слова "Комментарий" и нажмем кнопку **OK** (рис. 4.136).

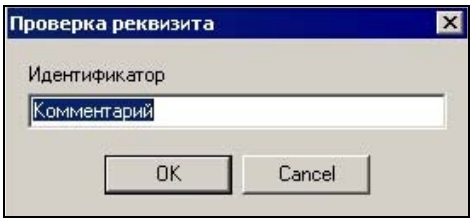



**Рис. 4.136.** Запрос на изменение идентификатора тестируемого реквизита

На экране появится сообщение о результате тестирования реквизита (рис. 4.137).

Повторим ту же операцию, но для некорректного (несуществующего) реквизита (рис. 4.138).

На экране появится сообщение о результате тестирования несуществующего реквизита (рис. 4.139).

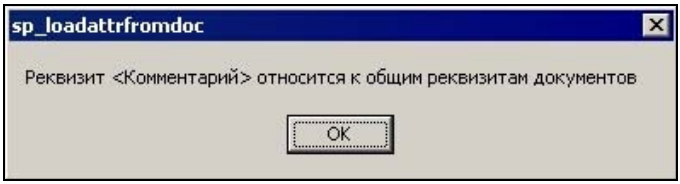



**Рис. 4.137.** Результат тестирования реквизита документа

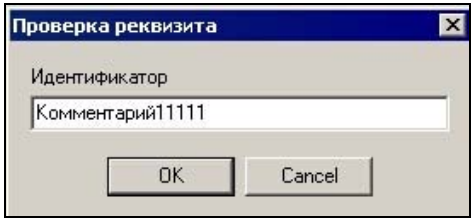



**Рис. 4.138.** Ввод заведомо несуществующего в документе реквизита

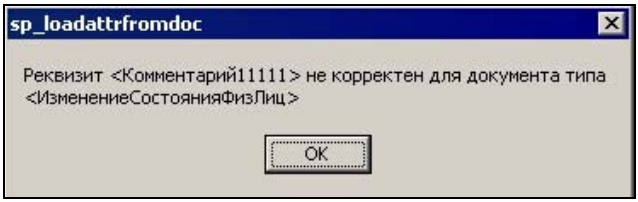



**Рис. 4.139.** Результат тестирования несуществующего реквизита документа

Точно так же протестируем реквизиты: `ПриоритетДокумента` и `НовоеСостояниеФизЛица`.

Убедимся в том, что разработанная нами функция `YesAttrInDoc_1C` работает корректно. Вернемся к нашему проекту (sp_1Cv7_Viewer_via_OLE.DPR) и доработаем его таким образом, чтобы можно было читать информацию из документов.

***Примечание***

Исходные тексты этого проекта есть на сопроводительном компакт-диске книги в папке \Part_1\Chapter_04\sp_1Cv7_Viewer_via_OLE.pas\04.

Для этого добавим новую вкладку и разместим на ней соответствующие компоненты (рис. 4.140). Здесь все аналогично вкладке **Справочники**.

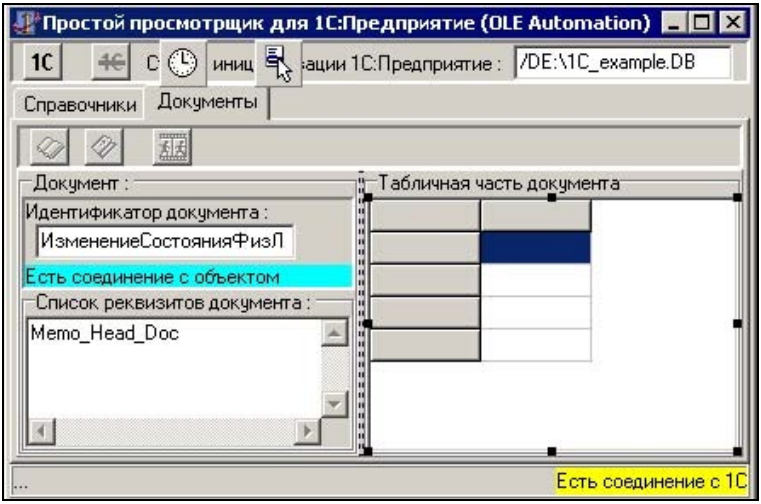



**Рис. 4.140.** Доработка формы проекта для чтения информации из документов

Вопросы соединения с объектом типа `Документ` также были рассмотрены ранее. Поэтому уделим внимание именно процессу считывания информации из документов. Поставим перед собой следующую задачу: научиться выбирать нужный документ из списка документов системы 1С:Предприятие в интерактивном режиме и считывать информацию из выбранного документа (как "шапку" документа, так и его табличную часть).

Рассмотрим первую часть задачи — выбор нужного документа в интерактивном режиме. Начнем с самого простого варианта. Будем использовать метод документа `Выбрать` (или `Choose`). Этот метод вызывает диалоговое окно для выбора документа и затем позиционирует выбранный документ в качестве текущего (данный метод может использоваться только для объектов, созданных функцией `СоздатьОбъект`). Синтаксис:

`Выбрать(`*`<Подсказка>`*`, `*`<ФормаЖурнала>`*`, `*`<КомуПодч>`*`)`

Параметры: *`<Подсказка>`* — текст заголовка окна диалога ввода; *`<Форма Журнала>`* — строковое выражение идентификатора той формы журнала документа (как он объявлен в Конфигураторе), которая должна использоваться

для выбора (если это значение пустое, то используется форма журнала по умолчанию); *<КомуПодч>* (необязательный) — используется при открытии выбора по журналу подчиненных документов, т. е. когда второй параметр *<ФормаЖурнала>* имеет значение `Журнал.Подчиненные` (в данном случае в этом параметре передается документ-владелец, по которому следует построить журнал подчиненных документов). Возвращаемое значение (число): 1 — если документ выбран; 0 — если не выбран.

Посмотрим, каким образом можно использовать метод `Выбрать` документа для наших целей. Простейший пример вызова метода `Выбрать` приведен в листинге 4.37.

**Листинг 4.37. Вызов метода `Выбрать` документа**

```
Var
  ResOpen: integer;
  ole_1C_Doc, // Ссылка на документ
  ole_1C: OleVariant; // Ссылка на 1С:Предприятие
begin
 // Вызываем 1С:Предприятие,как сервер OLE Automation
 ole_1C:= CreateOleObject('V77.Application');
 if not VarIsEmpty(ole_1C) then begin
   // Инициализация программы
   ResOpen:=ole_1C.Initialize(ole_1C.RMTrade, ...);
   if ResOpen<0 then begin // Инициализация — успешно
     // Создаем ссылку на объект типа "Документ"
ole_1C_Doc:=ole_1C.CreateObject('Документ.Здесь_Идентификатор_Документа');
     if not VarIsEmpty(ole_1C_doc) then begin
       // Вызываем метод "Выбрать" документа
       if ole_1C_doc.Выбрать('Выбрать документ')>0 then begin
           ...
           ...
           ...
       end;
     end;
   end;
 end;
end;
```

Как видите, здесь все предельно просто.

В данном случае мы вызываем метод `Выбрать` только с одним параметром — подсказкой для пользователя. На рис. 4.141 представлена форма выбора документов, вызванная при выполнении этого метода. В данном случае список пуст, потому что за указанный период (11.07.2006—11.07.2006) не оказалось ни одного документа.

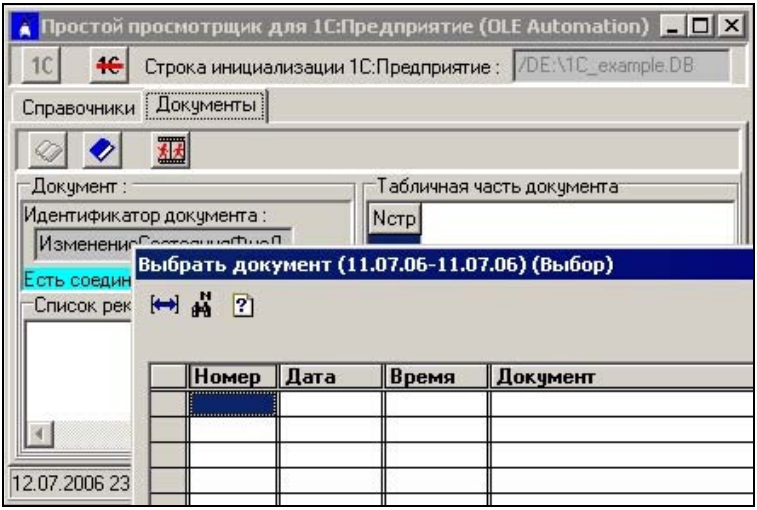



**Рис. 4.141.** Вызов метода `Выбрать` документа

Для того чтобы найти нужный нам документ, надо указать корректный период (см. рис. 4.143) в форме выбора документов, нажав соответствующую кнопку (см. рис. 4.141 и 4.142).

На экране появится форма ввода нового диапазона дат (рис. 4.143).

**Рис. 4.142.** Кнопка вызова формы для ввода периода отбора документов

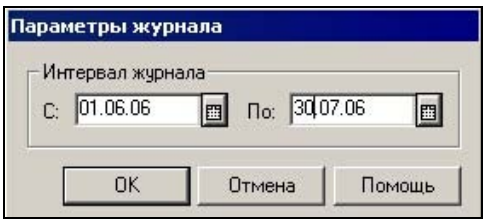



**Рис. 4.143.** Ввод периода, за который нужно отобрать документы

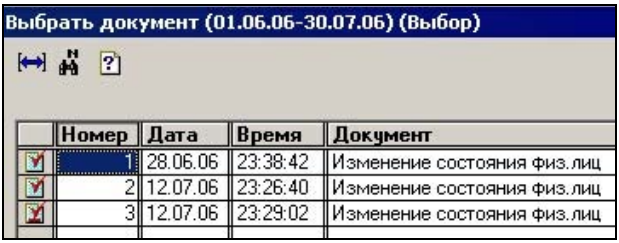



**Рис. 4.144.** Список документов для выбора при корректно указанном периоде отбора документов

После ввода нового диапазона дат в списке появятся соответствующие документы (рис. 4.144).

***Примечание***

Окно программы 1С:Предприятие (как сервера OLE Automation) по умолчанию остается невидимым. Поэтому диалоговое окно выбора документов (см. рис. 4.141) может быть скрыто другими окнами различных приложений, запущенных в данный момент на компьютере. В этом случае для "поиска" диалогового окна можно использовать сочетание клавиш <Alt>+<Tab>.

На рис. 4.50 показана пиктограмма, которую нужно выбрать для того, чтобы диалоговое окно выбора документов появилось поверх остальных окон.

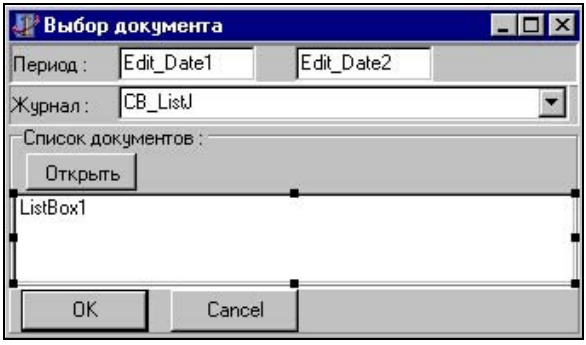



**Рис. 4.145.** Форма диалога для выбора документов

Для решения нашей задачи вполне подходит метод `Выбрать` документа. Но при решении практических задач этого может оказаться недостаточно, хотя бы потому, что трудно будет объяснить конечному пользователю, почему он должен искать окно формы выбора документов непонятно где и как. Поэтому попробуем разработать свою простую форму выбора документов. Пользователь (при вызове этой формы) должен иметь возможность указать период,

список документов за который он желал бы видеть, а также журнал, ограничивающий выборку документов. Для этого создадим новую форму диалога с именем `Dlg_Sel_Doc` (рис. 4.145).

В табл. 4.6 описаны основные компоненты этой формы.

***Таблица 4.6.*** *Назначение основных компонентов формы выбора документов*

| Компонент | Тип | Назначение |
|---|---|---|
| `Edit_Date1` | `TEdit` | Дата начала периода |
| `Edit_Date2` | `TEdit` | Дата окончания периода |
| `CB_ListJ` | `TComboBox` | Список журналов документов |
| `ListBox1` | `TListBox` | Список документов заданного журнала за заданный период |
| `SpeedButton1` | `TSpeedButton` | Кнопка **Открыть**. По нажатию этой кнопки заполняется список документов (`ListBox1`) с учетом фильтра, определенного периодом и заданным журналом документов |

Первое, что нужно сделать, — это создать список идентификаторов журналов документов, хранящихся в базе данных 1С:Предприятие. В листинге 4.38 приведена часть текста метода `Execute` нашей формы, где решается эта задача.

**Листинг 4.38. Создание списка журналов документов, хранящихся в базе данных 1С:Предприятие**

```
CB_ListJ.Items.Clear;
CB_ListJ.Items.ADD('Все документы');
CountJ:=Form1.ole_1C.Метаданные.Журнал*1;
if CountJ>0 then begin
 iJ:=0;
 while iJ<CountJ do
 begin
 iJ:=iJ+1;
 Sx:=Trim(Form1.ole_1C.Метаданные.Журнал(iJ*1).Идентификатор);
 CB_ListJ.Items.ADD(Sx);
 Application.ProcessMessages;
 end;
 CB_ListJ.ItemIndex:=0;
end;
```

Поясним листинг 4.38. Здесь, как и ранее, мы обращаемся к методам объекта `Метаданные`.

В строке:

```
CountJ:=Form1.ole_1C.Метаданные.Журнал*1;
```

вызывается метод метаданных — `Журнал` без параметров. В этом случае возвращается общее количество журналов, хранящихся в базе данных 1С:Предприятие.

В строке:

```
Sx:=Trim(Form1.ole_1C.Метаданные.Журнал(iJ*1).Идентификатор);
```

метод `Журнал` вызывается с параметром типа `Integer`. При этом возвращается ссылка на журнал как объект, порядковый номер которого соответствует значению параметра. Таким образом можно обратиться к свойству `Идентификатор` этого объекта.

Все остальное — тривиально.

Следующий шаг: создать фильтр, ограничивающий список документов, согласно заданным условиям (период и журнал).

В листинге 4.39 приведена часть текста обработчика события "нажатие кнопки **Открыть** нашей формы", где решается эта задача.

**Листинг 4.39. Условия выборки документов согласно указанному фильтру**

```
// Получаем ссылку на объект типа "Документ" неопределенного вида
OleDoc:= Form1.ole_1C.CreateObject('Документ');
Yes:=0; // Признак того, что выборка документов произведена успешно
if CB_ListJ.ItemIndex>0 then begin
   // Если указан конкретный журнал — вызываем
   // соответствующий метод документа:
   // "ИспользоватьЖурнал".
   // В противном случае фильтрация документов
   // будет производиться без учета журнала.
   OleDoc.ИспользоватьЖурнал(''+CB_ListJ.Items[CB_ListJ.ItemIndex]);
end;
if (length(Edit_Date1.Text)>0) and (length(Edit_Date2.Text)>0) then begin
   // Если период дат указан корректно, вызываем
   // метод "ВыбратьДокументы" документа и передаем
   // в качестве параметров обе даты,
   // ограничивающие период выборки документов
```

```
   if OleDoc.ВыбратьДокументы(''+Edit_Date1.Text, ''+Edit_Date2.Text)>0
   then begin
      Yes:=1;
   end;
end
else begin
   // Если период дат не указан, вызываем
   // метод "ВыбратьДокументы" документа без параметров,
   // тем самым снимая ограничения в выборке документов
   // по периоду
   if OleDoc.ВыбратьДокументы>0 then begin
      Yes:=2;
   end;
end;
```

Поясним листинг 4.39.

Сначала создается ссылка на объект типа `Документ` неопределенного вида (без идентификатора):

```
OleDoc:= Form1.ole_1C.CreateObject('Документ');
```

Это связано с тем, что мы не знаем заранее, какие именно условия укажет пользователь для фильтрации документов. То есть в списке могут оказаться разные документы.

Если пользователь указал для фильтрации конкретный журнал из сформированного ранее списка (листинг 4.38), то вызывается метод `ИспользоватьЖурнал` документа, где в качестве параметра передается идентификатор выбранного журнала:

```
OleDoc.ИспользоватьЖурнал(''+CB_ListJ.Items[CB_ListJ.ItemIndex]);
```

Если пользователь указал корректный диапазон дат, вызывается метод `ВыбратьДокументы` и в качестве параметров ему передаются обе даты, ограничивающие выборку документов соответствующим периодом:

```
if OleDoc.ВыбратьДокументы(''+Edit_Date1.Text,''+Edit_Date2.Text)>0 then
```

Если пользователь не указал диапазон дат, вызывается метод `ВыбратьДокументы`, но без параметров:

```
if OleDoc.ВыбратьДокументы>0 then begin
```

Вот, собственно, и все.

> ***Примечание***
>
> Более подробно методы `ИспользоватьЖурнал` и `ВыбратьДокументы` документа описаны в специальной литературе по системе 1С:Предприятие.

Следующий шаг: сформировать список документов на основе произведенной выборки (см. листинг 4.39). Часть текста обработчика события "нажатие кнопки **Открыть** нашей формы", где решается эта задача, приведена в листинге 4.40.

**Листинг 4.40. Формирование списка документов**

```
if Yes>0 then begin
   // Формируем список, выбирая документы в цикле
   while OleDoc.ПолучитьДокумент > 0 do
    begin
      // Считываем ссылку на "ТекущийДокумент"
      OleBuf:=OleDoc.ТекущийДокумент;
      // Формируем (как сами пожелаем) представление документа
      // в нашем списке документов
      Sx:= Trim(OleBuf.Вид)
              + '  N '
              + Trim(oleBuf.НомерДок)
              + ' ['
              + Trim(oleBuf.ДатаДок)
              + ']';
      // Вносим документ в наш список
      ListBox1.Items.ADD(Sx);
      Application.ProcessMessages;
   end;
end;
```

Поясним текст листинга 4.40.

Поскольку выборка документов была уже произведена (листинг 4.39), считываем каждый документ в цикле, используя метод `ПолучитьДокумент`:

```
while OleDoc.ПолучитьДокумент > 0 do
```

Чтобы получить ссылку на документ, нужно вызвать метод `ТекущийДокумент`:

```
OleBuf:=OleDoc.ТекущийДокумент;
```

Далее формируем представление документа в нашем списке так, как сами пожелаем. В данном случае (упрощенно) для формирования представления документа используются методы и свойства: `Вид`, `НомерДок`, `ДатаДок`.

Метод `ТекущийДокумент` (или `CurrentDocument`) документа позволяет определить значение текущего документа. Синтаксис:

```
ТекущийДокумент()
```

Метод `ТекущийДокумент` возвращает значение позиционированного текущего документа (в целом как объекта). Данный метод применяется, например, если нужно документ передать как параметр в вызове какого-либо метода или присвоить какому-либо реквизиту.

На рис. 4.146 представлена форма (в режиме времени выполнения, естественно), когда сформирован список журналов (в данном случае журнал один).

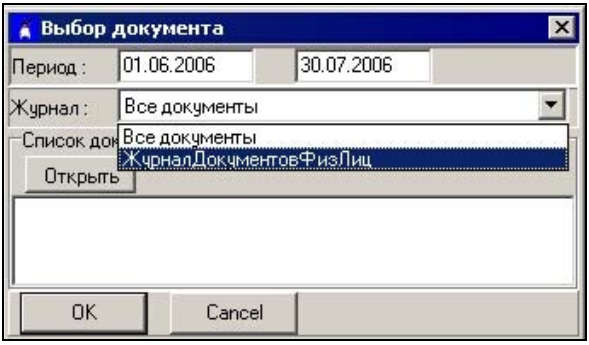



**Рис. 4.146.** Выбор журнала для фильтрации документов

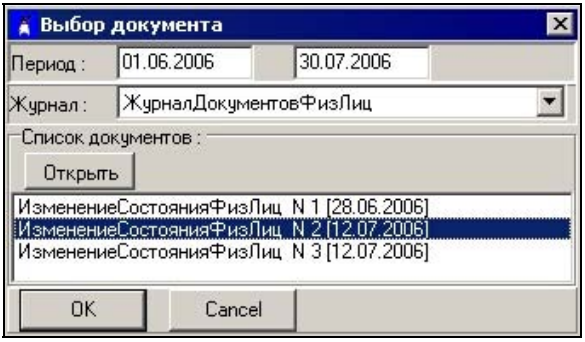



**Рис. 4.147.** Список документов, сформированный при нажатии кнопки **Открыть**

На рис. 4.147 представлен список документов, сформированный для выбора по указанным выше условиям фильтрации (рис. 4.146).

***Примечание***

На момент формирования списка документов (см. рис. 4.147) в базе данных 1С:Предприятие было 3 документа и один журнал.

Теперь, когда список документов сформирован, мы можем из выбранной строки выделить идентификатор документа (первое слово) и номер документа

(третье слово). Часть текста метода `Execute` нашей формы, где решается эта задача, приведена в листинге 4.41.

**Листинг 4.41. Обработка выбранной строки с представлением документа**

```
if ShowModal = mrOk then begin
   // Форма закрывается по нажатию кнопки OK
   if ListBox1.ItemIndex>=0 then begin
      // Если в списке документов выбрана строка, то
      // выделяем из нее идентификатор и номер документа
      Sx:=ListBox1.Items[ListBox1.ItemIndex];
      // Выделяем из строки идентификатор документа (1-е слово)
      idDoc := sp_GetWordFromString(' ][',Sx,1);
      // Выделяем из строки номер документа (3-е слово)
      NumDoc:= sp_GetWordFromString(' ][',Sx,3);
      // Выделяем из строки дату документа (4-е слово)
      DateDoc:= sp_GetWordFromString(' ][',Sx,4);
      if (length(idDoc)>0) and (length(NumDoc)>0) then begin
         Result:=TRUE;
      end;
   end
   else begin
      ShowMessage('Внимание.Ни один из документов не выбран');
   end;
end;
```

Поясним листинг 4.41. Текст достаточно прост. Единственное, что следует отметить, — в данном случае для идентификации документа в базе данных 1С:Предприятие (в нашей конфигурации) мы ориентируемся на идентификатор документа, его номер и дату, предполагая, что эти два свойства в совокупности являются уникальными для каждого документа.

Для выделения из строки слова по номеру используется функция `sp_GetWordFromString`. Текст этой функции приведен в файле \sp_Lib\sp_Lib_Delphi.PAS на сопроводительном компакт-диске книги.

Теперь, когда нужный документ выбран (определен его идентификатор и номер), можно прочитать из него информацию. Рассмотрим этот вопрос более подробно.

В нашем случае, чтобы получить ссылку на выбранный документ как на объект, нужно создать объект типа `Документ.Идентфикатор_Выбранного_Документа`:

```
ole_1C_Doc:=ole_1C.CreateObject('Документ.'+ Dlg_Sel_Doc.idDoc);
```

Этим самым мы определяем вид документов, среди которых нужно найти наш документ по номеру.

Затем, воспользовавшись методом документа `НайтиПоНомеру`, найдем (спозиционируем) сам документ:

```
Snum := Dlg_Sel_Doc.NumDoc;
Sdate:= Dlg_Sel_Doc.DateDoc;
if ole_1C_doc.НайтиПоНомеру(''+Snum,''+Sdate)>0 then begin
```

После этого можно прочитать значения общих свойств документа (не путать с общими реквизитами), такие как время и позиция документа, признак того, проведен ли этот документ, количество строк в табличной части документа и др. Пример считывания приведен в листинге 4.42.

**Листинг 4.42. Чтение общих свойств документа**

```
if ole_1C_doc.НайтиПоНомеру(''+Dlg_Sel_Doc.NumDoc)>0 then begin
  //---------------------------------------------
  //Прочитаем общие свойства документа
  Sx:=Trim(ole_1C_doc.НомерДок);
  Memo_Head_Doc.Lines.ADD('Документ N '+Sx);
  Sx:=Trim(ole_1C_doc.ДатаДок);
  Memo_Head_Doc.Lines.ADD('Дата: '+Sx);
  Sx:=Trim(ole_1C_doc.ПолучитьВремя);
  Memo_Head_Doc.Lines.ADD('Время: '+Sx);
  Sx:=Trim(ole_1C_doc.ПолучитьПозицию);
  Memo_Head_Doc.Lines.ADD('Позиция: '+Sx);
  Sx:=Trim(ole_1C_doc.Вид);
  Memo_Head_Doc.Lines.ADD('Вид: '+Sx);
  Sx:=Trim(ole_1C_doc.ПредставлениеВида);
  Memo_Head_Doc.Lines.ADD('Представление вида: '+Sx);
  CountLine:=ole_1C_doc.КоличествоСтрок*1;
  Memo_Head_Doc.Lines.ADD('Кол-во строк='+IntToStr(CountLine));
  if ole_1C_doc.Проведен>0 then begin
     Memo_Head_Doc.Lines.ADD('Проведен');
  end
```

```
  else begin
     Memo_Head_Doc.Lines.ADD('Не проведен');
  end;
  //---------------------------------------------
```

***Примечание***

Более подробно методы и свойства объекта типа `Документ`, используемые в листинге 4.42, описаны в специальной литературе по системе 1С:Предприятие.

Теперь можно считывать значения реквизитов документа (общих, "шапки" и табличной части).

Сначала прочитаем значения общих реквизитов документов для данного документа. На момент считывания информации наверняка неизвестно, какие именно общие реквизиты присутствуют в конфигурации системы. Поэтому нужно получить список идентификаторов общих реквизитов. Для этого воспользуемся функцией `Get_DocAttrCommon_to_List` (см. листинг 4.26).

Фрагмент исходного текста, решающего эту задачу, приведен в листинге 4.43.

**Листинг 4.43. Чтение значений общих реквизитов документов для данного документа**

```
ListBuf:= TStringList.Create;
TRY
  //---------------------------------------------
  // Прочитаем идентификаторы общих реквизитов документов
  Get_DocAttrCommon_to_List(ole_1C,ListBuf);
  if ListBuf.Count>0 then begin
     iAttr:= -1;
     while iAttr<(ListBuf.Count-1) do
      begin
       iAttr:=iAttr+1;
       idAttr:=Trim(ListBuf[iAttr]);
       if length(idAttr)>0 then begin
          // Выделим идентифкатор реквизита
          idAttr:=sp_GetWordFromString(' =',idAttr,1);
          Sx:='';
          // Прочитаем значение реквизита
          Sx:=Trim(ole_1C_doc.ПолучитьАтрибут(''+idAttr));
          Memo_Head_Doc.Lines.ADD(idAttr+'='+Sx);
```

```
      end;
    end;
  end;
  //---------------------------------------------
```

Для считывания значения реквизита по его идентификатору используется метод `ПолучитьАтрибут`.

В общем случае нельзя присваивать переменной типа `String` значение, возвращаемое методом `ПолучитьАтрибут`. Если реквизит имеет агрегатный тип, это приведет к ошибке. То есть необходимо определить тип значения и соответствующим образом корректно преобразовать его в строку. Этот вопрос был подробно рассмотрен в *разд. 4.1*.

В данном случае точно известно, что единственный общий реквизит документов `Комментарий` имеет тип `Строка`. Поэтому, чтобы не усложнять текст, все проверки и преобразования здесь отсутствуют. Если кто-либо пожелает использовать этот текст для доступа к информации из реальной базы данных 1С, то нужно либо закомментировать строку:

```
Sx:=Trim(ole_1C_doc.ПолучитьАтрибут(''+idAttr));
```

либо самостоятельно написать функцию, которая бы корректно преобразовывала значение, возвращаемое методом `ПолучитьАтрибут`, в тип `String`.

Для считывания значений реквизитов "шапки" и табличной части документа нужно получить ссылку на текущий документ (т. е. ссылку на спозиционированный документ):

```
OleTekDoc:=ole_1C_doc.ТекущийДокумент;
```

На момент считывания информации также неизвестно, какие именно реквизиты "шапки" присутствуют в документах каждого вида. Поэтому, как и в случае с общими реквизитами, сначала нужно получить список идентификаторов реквизитов "шапки" для документов выбранного вида. Для этого воспользуемся функцией `Get_DocAttrHead_to_List` (листинг 4.27). Фрагмент исходного кода, решающего эту задачу, приведен в листинге 4.44.

**Листинг 4.44. Чтение значений реквизитов шапки документа**

```
ListBuf:= TStringList.Create;
TRY
   // Получим ссылку на текущий (спозиционированный) документ
   OleTekDoc:=ole_1C_doc.ТекущийДокумент;
   //---------------------------------------------
   // Прочитаем реквизиты "шапки" документа
```

```
Get_DocAttrHead_to_List(ole_1C,OleTekDoc,ListBuf);
if ListBuf.Count>0 then begin
   iAttr:= -1;
   while iAttr<(ListBuf.Count-1) do
    begin
     iAttr:=iAttr+1;
     idAttr:=Trim(ListBuf[iAttr]);
     if length(idAttr)>0 then begin
        // Выделим идентифкатор реквизита
        idAttr:=sp_GetWordFromString(' =',idAttr,1);
        // Прочитаем значение реквизита
        Sx:=Trim(Get_ValueAttr_from_Doc_str(ole_1C,OleTekDoc,idAttr));
        Memo_Head_Doc.Lines.ADD(idAttr+'='+Sx);
     end;
   end;
end;
//---------------------------------------------
```

Среди реквизитов "шапки" документов вида `ИзменениеСостоянияФизЛиц` присутствуют объекты агрегатного типа (`Перечисление.ПриоритетДокумента`). Поэтому для считывания значений реквизитов и преобразования их в тип `String` мы воспользовались специальной функцией `Get_ValueAttr_from_Doc_str`. Полный текст этой функции приведен на сопроводительном компакт-диске книги в файле \sp_Lib\sp_Lib_Delphi.PAS.

> ***Примечание***
>
> Принципы преобразования значений агрегатного типа в тип `String` были подробно рассмотрены в *разд. 4.1*. Для документов этот процесс аналогичен. Поэтому останавливаться на нем не будем.

Теперь посмотрим, как считываются реквизиты табличной части документа (листинг 4.45). Подход здесь такой же, но с учетом того, что считываются значения реквизитов *строк* документа. То есть прежде чем считывать значение реквизита табличной части документа, нужно "спозиционировать" соответствующую строку документа.

**Листинг 4.45. Чтение значений реквизитов табличной части документа**

```
ListBuf:= TStringList.Create;
TRY
  // Прочитаем идентификаторы реквизитов табличной
```

```
// части документов заданного вида
Get_DocAttrTable_to_List(ole_1C,OleTekDoc,ListBuf);
if ListBuf.Count>0 then begin
   //=========================================
   // Выберем строки документа,
   // прочитаем значения реквизитов табличной части
   // для каждой строки
   if ole_1C_doc.ВыбратьСтроки>0 then begin
      // Переберем все строки документа в цикле
      while ole_1C_doc.ПолучитьСтроку>0 do
       begin
         // Номер текущей (спозиционированной) строки документа
         iLine:=ole_1C_doc.НомерСтроки*1;
         //----------------------------
         // Прочитаем значения реквизитов текущей строки
         iAttr:= -1;
         while iAttr<(ListBuf.Count-1) do
          begin
           iAttr:=iAttr+1;
           idAttr:=Trim(ListBuf[iAttr]);
           // Выделим идентифкатор реквизита
           idAttr:=sp_GetWordFromString(' =',idAttr,1);
            // Прочитаем значение реквизита
     Sx:=Trim(Get_ValueAttr_from_Doc_str(ole_1C,ole_1C_doc,idAttr));
         end;
         //----------------------------
         Application.ProcessMessages;
      end;
   end;
   //=========================================
```

Текст этого фрагмента достаточно прост и не требует разъяснений.

***Примечание***

Более подробно методы `ВыбратьСтроки`, `ПолучитьСтроку` и свойство `НомерСтроки` документа описаны в специальной литературе по системе 1С:Предприятие.

Для перебора строк документа можно также использовать метод `Получить СтрокуПоНомеру` документа. Фрагмент функции, решающей эту задачу, представлен в листинге 4.46.

**Листинг 4.46. Чтение значений реквизитов табличной части документа**

```
// Прочитаем количество строк в табличной части
// данного документа
CountLine:=ole_1C_doc.КоличествоСтрок*1;
ListBuf:= TStringList.Create;
TRY
  // Прочитаем идентификаторы реквизитов табличной
  // части документов заданного вида
  Get_DocAttrTable_to_List(ole_1C,OleTekDoc,ListBuf);
  if ListBuf.Count>0 then begin
     //=========================================
     // Выберем строки документа,
     // прочитаем значения реквизитов табличной части
     // для каждой строки
     if ole_1C_doc.ВыбратьСтроки>0 then begin
        iLine:=0;
        while iLine<CountLine do
         begin
          iLine:=iLine+1;
          if ole_1C_doc.ПолучитьСтрокуПоНомеру(iLine*1)*1>0 then begin
             //----------------------------
             // Прочитаем значения реквизитов текущей строки
             iAttr:= -1;
             while iAttr<(ListBuf.Count-1) do
              begin
               iAttr:=iAttr+1;
               idAttr:=Trim(ListBuf[iAttr]);
               // Выделим идентифкатор реквизита
               idAttr:=sp_GetWordFromString(' =',idAttr,1);
               // Прочитаем значение реквизита
        Sx:=Trim(Get_ValueAttr_from_Doc_str(ole_1C,ole_1C_doc,idAttr));
             end;
             //----------------------------
          end;
```

```
        Application.ProcessMessages;
      end;
    end;
    //=========================================
```

Здесь для перебора всех строк документа в цикле используются методы `КоличествоСтрок` и `ПолучитьСтрокуПоНомеру`. Все остальное аналогично предыдущему примеру.

На рис. 4.148 представлен результат считывания информации из документа с номером 1.

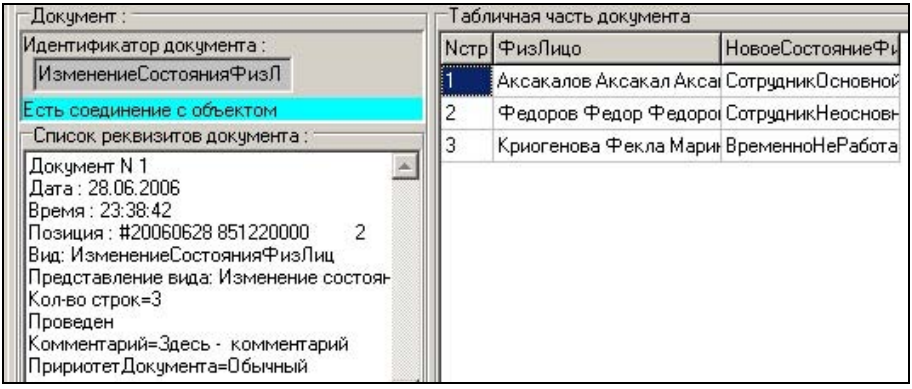



**Рис. 4.148.** Результат считывания информации из документа `ИзменениеСостоянияФизЛиц` с номером 1

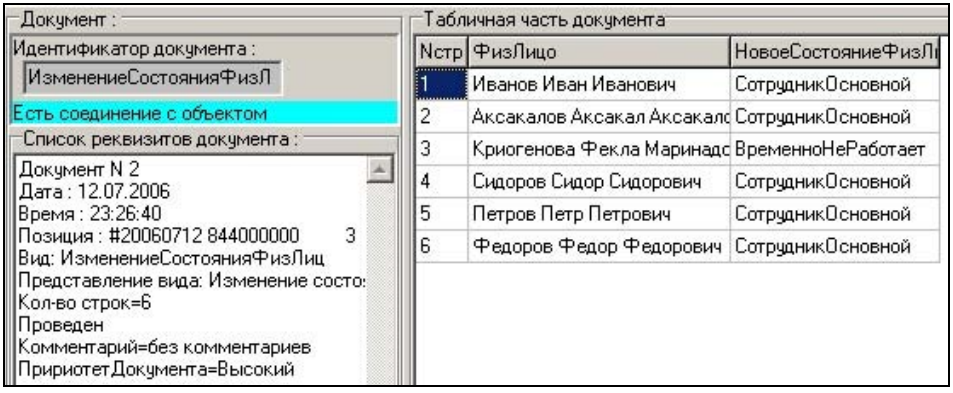



**Рис. 4.149.** Результат считывания информации из документа `ИзменениеСостоянияФизЛиц` с номером 2

Можно сравнить этот результат с той информацией, которую мы внесли в документ в программе 1С:Предприятие (см. рис. 4.113).

На рис. 4.149 представлен результат считывания информации из документа с номером 2.

**Примечание**

На сопроводительном компакт-диске книги в папке \Part_1\Chapter_04\sp_1Cv7_Viewer_via_OLE.PAS\05 есть пример, иллюстрирующий то, что мы здесь вкратце разобрали.

**Примечание**

Напомню, что представленные выше примеры исходных текстов — это упрощенная иллюстрация к рассматриваемым вопросам. Назначение этих примеров — дать представление о возможном подходе при решении задач информационного обмена с системой 1С:Предприятие. При работе с реальными базами 1С может потребоваться их дальнейшая доработка.

# 4.3. Считывание информации из регистров

Регистры — это средство накопления сводной информации. Существуют два типа регистров:

- *регистр остатков* — предназначен для ведения остатков "ресурсов" на момент времени с привязкой к набору "измерений" регистра;
- *оборотный регистр* — предназначен для подсчета оборота "ресурсов" за заданный интервал времени с привязкой к набору "измерений" регистра.

Тип регистра задается при настройке конфигурирации системы.

*Ресурсы* регистра — это информация, по которой ведется численный учет (например, количество детей, сумма выплат и т. д.).

Регистр можно представить в виде многомерного "куба": в таком представлении *измерения* регистра — это оси координат, на пересечении которых регистр хранит конкретные значения ресурсов.

Для дополнительной информации о движении и фильтрации по регистру используются *реквизиты* регистра.

**Примечание**

К регистрам нет непосредственного доступа через интерфейс пользователя системы 1С:Предприятие.

**Примечание**

Более подробную информацию о регистрах можно найти в специальной литературе по системе 1С:Предприятие.

Сделаем очередные изменения в нашей учебной конфигурации. Для этого откроем дерево конфигурации и введем регистр `УчетИзмененийСостояния-ФизЛиц` (рис. 4.150).

**Рис. 4.150.** Новый регистр в конфигурации системы

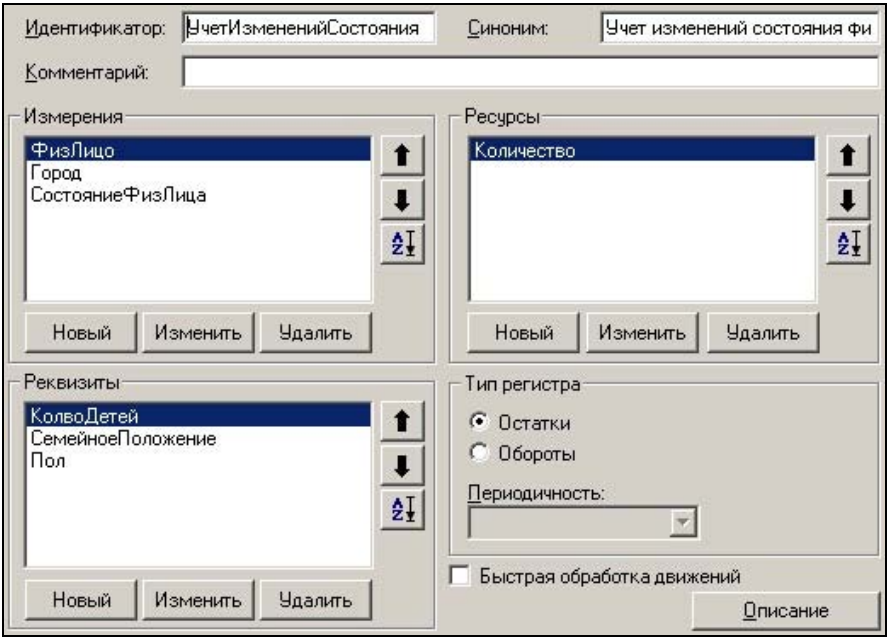



**Рис. 4.151.** Диалоговое окно настройки регистра

В диалоговом окне настройки регистра (рис. 4.151) создадим структуру этого регистра.

Измерения:

- `ФизЛицо` — `Справочник.ФизЛица`
- `Город` — `Справочник.ГородаСтранМира`
- `СостояниеФизЛица` — `Перечисление.СостояниеФизлица`

Ресурсы:

- `Количество` — `Число`

Реквизиты:

- КолвоДетей — Число
- СемейноеПоложение — Перечисление.СемейноеПоложение
- Пол — Перечисление.Пол

На рис. 4.152 представлена структура регистра в дереве конфигурации.

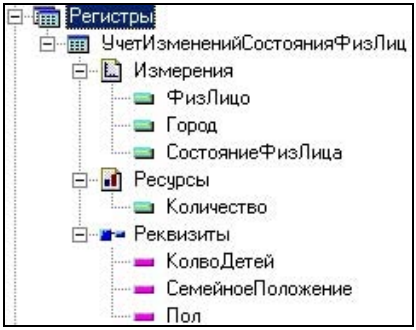



**Рис. 4.152.** Структура регистра в дереве конфигурации системы

Откроем окно настройки свойств документа ИзменениеСостоянияФизЛиц (см. рис. 4.107) и внесем изменения, как показано на рис. 4.153.

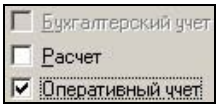



**Рис. 4.153**. Изменения в настройке документа вида ИзменениеСостоянияФизЛиц

Эти изменения позволят при проведении документа делать движения по регистрам оперативного учета.

Сделаем соответствующие изменения в модуле документа (листинг 4.47).

**Листинг 4.47. Процедура обработки проведения документа ИзменениеСостоянияФизЛиц**

```
Процедура ОбработкаПроведения()
// Здесь программист должен написать программный код,
// реализующий алгоритм проведения документов данного вида.
ВыбратьСтроки();
Пока ПолучитьСтроку() = 1 Цикл
```

```
  СтарСостФизлица = "";
  Пол = "";
  Город = "";
  СемейноеПоложение = "";
  Если ФизЛицо.Выбран()>0 тогда
СтарСостФизлица=ФизЛицо.ПолучитьАтрибут(""+"СостояниеФизлица").Получить(
  ДатаДок);
КолвоДетей=ЧИСЛО(ФизЛицо.ПолучитьАтрибут(""+"КолвоДетей").Получить(
  ДатаДок));
    Пол = ФизЛицо.Пол;
    Город = ФизЛицо.ПолучитьАтрибут(""+"ГородПроживания");
    СемейноеПоложение = ФизЛицо.СемейноеПоложение;
    //===================
    // Записать новые значения периодических реквизитов
    УстановитьРеквизитСправочника(
                                  ФизЛицо,
                                  "СостояниеФизлица",
                                  НовоеСостояниеФизЛица,
                                  ДатаДок
                                  );
    //===================
    //===================
    // Записать движения по регистру УчетИзмененийСостоянияФизЛиц
    //------------------
    // Расход по старому состоянию физ.лица
    // Измерения
    Регистр.УчетИзмененийСостоянияФизЛиц.ФизЛицо = ФизЛицо;
    Регистр.УчетИзмененийСостоянияФизЛиц.Город = Город;
Регистр.УчетИзмененийСостоянияФизЛиц.СостояниеФизЛица = СтарСостФизлица;
    // Реквизиты
    Регистр.УчетИзмененийСостоянияФизЛиц.СемейноеПоложение =
      СемейноеПоложение;
    Регистр.УчетИзмененийСостоянияФизЛиц.Пол = Пол;
    Регистр.УчетИзмененийСостоянияФизЛиц.КолвоДетей = КолвоДетей;
    // Ресурсы
    Регистр.УчетИзмененийСостоянияФизЛиц.Количество = 1;
    // Движение: Расход
    Регистр.УчетИзмененийСостоянияФизЛиц.ДвижениеРасходВыполнить();
    //------------------
```

```
    //------------------
    // Приход по новому состоянию физ.лица
    // Измерения
    Регистр.УчетИзмененийСостоянияФизЛиц.ФизЛицо = ФизЛицо;
    Регистр.УчетИзмененийСостоянияФизЛиц.Город = Город;
    Регистр.УчетИзмененийСостоянияФизЛиц.СостояниеФизЛица =
      НовоеСостояниеФизлица;
    // Реквизиты
    Регистр.УчетИзмененийСостоянияФизЛиц.СемейноеПоложение =
      СемейноеПоложение;
    Регистр.УчетИзмененийСостоянияФизЛиц.Пол = Пол;
    Регистр.УчетИзмененийСостоянияФизЛиц.КолвоДетей = КолвоДетей;
    // Ресурсы
    Регистр.УчетИзмененийСостоянияФизЛиц.Количество = 1;
    // Движение: Приход
    Регистр.УчетИзмененийСостоянияФизЛиц.ДвижениеПриходВыполнить();
    //------------------
    //===================
  КонецЕсли;
КонецЦикла;
КонецПроцедуры
```

Если сравнить текст листинга 4.47 с текстом листинга 4.23, то видно, что добавлены строки, обеспечивающие запись движений по регистру `УчетИзмененийСостоянияФизЛиц`.

Здесь для каждой строки документа создаются две записи в регистре: расход и приход (соответственно, методы регистров `ДвижениеРасходВыполнить` и `ДвижениеПриходВыполнить`). В данном случае расход производится по "старому" значению реквизита `СостояниеФизлица` справочника `ФизЛица`, а приход — по "новому" значению этого реквизита.

> ***Примечание***
>
> Более подробно регистры и механизм проведения документов описаны в специальной литературе по системе 1С:Предприятие.

Теперь нужно сохранить изменения в конфигурации, запустить программу 1С:Предприятие и перепровести все документы вида `ИзменениеСостояниеФизЛиц`.

Чтобы посмотреть движения по регистру, нужно в списке документов выбрать нужный документ (в нашем случае документ N 1), вызвать всплывающее меню и выбрать пункт **Движения документа** (см. рис. 4.116). На экране появится диалоговое окно для выбора вида движений, созданных данным документом (рис. 4.154).

**Рис. 4.154.** Список видов движений, созданных данным документом

Если сравнить рис. 4.154 и 4.117, то видно, что появился новый вид движений по регистру `УчетИзмененийСостоянияФизЛиц`.

На рис. 4.155 представлены все записи движения по этому регистру, созданные данным документом в результате его проведения.

| | N | ФизЛицо | Город | Состояние физ.лица | Количе... | Количество де... | Семейное положение |
|---|---|---|---|---|---|---|---|
| − | 1 | Аксакалов Аксакал | Киев | Сотрудник по неосновно | 1 | 3 | Разведен (разведена) |
| + | 1 | Аксакалов Аксакал | Киев | Сотрудник по основному | 1 | 3 | Разведен (разведена) |
| − | 2 | Федоров Федор Ф | Киев | Временно не работает | 1 | 2 | Разошелся (разошлась) |
| + | 2 | Федоров Федор Ф | Киев | Сотрудник по неосновно | 1 | 2 | Разошелся (разошлась) |
| − | 3 | Криогенова Фекла | Гомель | Сотрудник по основному | 1 | 1 | Никогда не состоял (не состо |
| + | 3 | Криогенова Фекла | Гомель | Временно не работает | 1 | 1 | Никогда не состоял (не состо |

**Рис. 4.155.** Записи движения по регистру `УчетИзмененийСостоянияФизЛиц`, созданные документом N 1

Созданные нами изменения в конфигурации отражены в словаре данных (файл 1Cv7.DD, рис. 4.156 и 4.157).

```
#==============================================================================
#==TABLE no 13     : Регистр УчетИзмененийСостоянияФизЛиц
# Name     |Descr                          |Type[A/S/U]|DBTableName|ReUsable
T=RG52     |Регистр УчетИзмененийСостояния|A          |RG52       |1
#-----Fields-------
# Name       |Descr                |Type|Length|Precision
F=PERIOD     |Period Registr       |D   |8     |0
F=SP53       |(P)ФизЛицо           |C   |9     |0
F=SP55       |(P)Город             |C   |9     |0
F=SP64       |(P)СостояниеФизЛица  |C   |9     |0
F=SP60       |(P)Количество        |N   |11    |0
#----Indexes------
# Name      |Descr          |Unique|Indexed fields                                  |DBName
I=PROP      |PERIOD+PROP    |0     |PERIOD,SP53,SP55,SP64                           |PROP
#
```

**Рис. 4.156.** Регистр остатков `УчетИзмененийСостоянияФизЛиц`. Таблица RG52.DBF. Итоги по регистру с учетом измерений

```
#==============================================================================
#==TABLE no 14     : Регистр (Дв.) УчетИзмененийСостоянияФизЛиц
# Name     |Descr                          |Type[A/S/U]|DBTableName|ReUsable
T=RA52     |Регистр (Дв.) УчетИзмененийСос|A          |RA52       |1
#-----Fields-------
# Name       |Descr                |Type|Length|Precision
F=IDDOC      |ID Document's        |C   |9     |0
F=LINENO     |LineNo               |N   |4     |0
F=ACTNO      |Action No            |N   |6     |0
F=DEBKRED    |Flag Debet/Kredit    |N   |1     |0
F=SP53       |(P)ФизЛицо           |C   |9     |0
F=SP55       |(P)Город             |C   |9     |0
F=SP64       |(P)СостояниеФизЛица  |C   |9     |0
F=SP60       |(P)Количество        |N   |11    |0
F=SP63       |(P)КолвоДетей        |N   |4     |0
F=SP57       |(P)СемейноеПоложение|C   |9     |0
F=SP61       |(P)Пол               |C   |9     |0
#----Indexes------
# Name      |Descr          |Unique|Indexed fields                                |DBName
I=IDLINE    |of IDDOC+LineN|0     |IDDOC,LINENO,ACTNO                            |IDLINE
```

**Рис. 4.157.** Регистр остатков `УчетИзмененийСостоянияФизЛиц`. Таблица RA52.DBF. Движение по регистру с учетом измерений и реквизитов

Как видите, для каждого нового регистра остатков создаются две таблицы (два DBF-файла), одна таблица с префиксом RG, другая — с префиксом RA. Более подробно это рассматривается в *части II*.

Теперь рассмотрим способы считывания информации о регистрах и из регистров. Например, в листинге 4.48 представлен текст функции, позволяющей определить общее количество регистров в данной конфигурации.

**Листинг 4.48. Определение количества регистров в данной конфигурации**

```
function Get_CountOfRegs(handle1C: OleVariant): integer;
// Получение количества регистров в конфигурации:
// handle1C - ссылка на 1С,
// возвращаемое значение - количество регистров в конфигурации
begin
  Result:=0;
  if not VarIsEmpty(handle1C) then begin
     Result:=handle1C.Метаданные.Регистр*1;
  end;
end;
```

Как видите, здесь тот же принцип, что и при определении количества документов в конфигурации *(см. разд. 4.2)*.

Так же, как в случае с другими агрегатными типами данных (рассмотренными ранее), информацию из регистра можно получить, обратившись к свой-

ствам и методам регистра (как объекта). Для этого нужно знать идентификатор регистра. Прочитать список идентификаторов регистров, существующих в конфигурации, можно следующим способом (листинг 4.49).

**Листинг 4.49. Чтение списка идентифкаторов регистров в данной конфигурации**

```
function Get_Regs_to_List(handle1C: OleVariant; NrMode: integer;
                          List: TStrings): integer;
// Получение списка идентификаторов регистров в конфигурации:
// handle1C — ссылка на 1С,
// NrMode   — указывает тип регистров, которые нужно
// считывать в список:
//            1 - только регистры остатков,
//            2 - только оборотные регистры,
//            0 - все регистры.
// List     — здесь формируется список идентификаторов регистров.
//           Переменная List должна быть создана в вызывающем модуле.
// Возвращаемое значение: количество прочитанных регистров.
Var
  Sx: string;
  OleBuf: OleVariant;
  Yes: byte;
  WhatIsIt,
  iReg,
  CountReg: integer; // Количество регистров в кофигурации
begin
  Result:=0;
  if List<>NIL then begin
     List.Clear;
     if not VarIsEmpty(handle1C) then begin
       // Определяем количество регистров в конфигурации
       CountReg:=handle1C.Метаданные.Регистр*1;
       // Организуем цикл по количеству регистров
       for iReg:=1 to CountReg do
        begin
          WhatIsIt:=0;
          // Получаем ссылку на объект метаданных "регистр"
          OleBuf:=handle1C.Метаданные.Регистр(iReg*1);
          // Считываем тип регистра
```

```
          Sx:=AnsiUpperCase(Trim(OleBuf.ТипРегистра));
          if Sx=AnsiUpperCase('Остатки') then WhatIsIt:=1;
          if Sx=AnsiUpperCase('Обороты') then WhatIsIt:=2;
          //-----------------------
          // Проверяем, нужно ли добавить этот регистр в список
          Yes:=1;
          if NrMode>0 then begin
             Yes:=0;
             if WhatIsIt=NrMode then Yes:=1;
          end;
          //-----------------------
          if Yes>0 then begin
             // Считываем идентифкатор регистра
             Sx:=Trim(OleBuf.Идентификатор);
             // Добавляем идентификатор в список
             List.ADD(Sx);
          end;
       end;
     end;
     Result:=List.Count;
  end;
end;
```

Здесь все достаточно тривиально. Единственное, что следует отметить: метод `ТипРегистра` (объекта `Регистр` метаданных) возвращает одно из строковых значений (в зависимости от типа регистра, указанного в конфигураторе):

- `"Остатки"` — если это регистр остатков;
- `"Обороты"` — если это оборотный регистр.

На рис. 4.158 и 4.159 представлен результат использования функции `Get_Regs_to_List`.

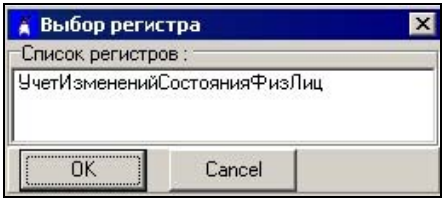



**Рис. 4.158.** Список идентификаторов регистров в "нашей" конфигурации

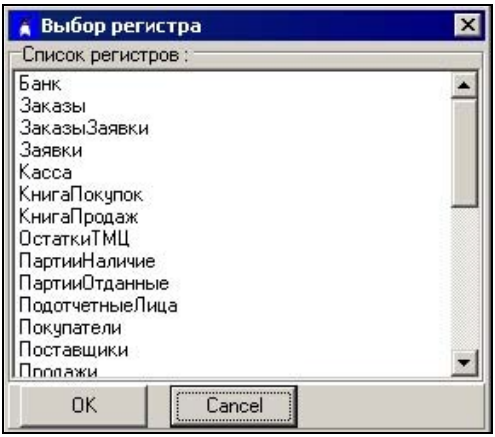



**Рис. 4.159.** Список идентификаторов регистров в реальной конфигурации

Теперь рассмотрим функцию, позволяющую прочитать количество измерений заданного регистра (листинг 4.50).

**Листинг 4.50. Функция для определения количества измерений заданного регистра**

```
function Get_CountMeasureOfRegs(handle1C: OleVariant; idReg: string):
integer;
// Получение количества измерений регистра:
// handle1C — ссылка на 1С,
// idReg — идентификатор регистра.
// Возвращаемое значение: количество измерений.
begin
  Result:=0;
  idReg:=Trim(idReg);
  if length(idReg)>0 then begin
     if not VarIsEmpty(handle1C) then begin
        Result:=handle1C.Метаданные.Регистр(''+idReg).Измерение*1;
     end;
  end;
end;
```

Здесь принцип тот же, что и при определении количества регистров в конфигурации (листинг 4.48). Единственное различие в том, что к регистрам (как объектам) нужно обращаться в контексте метатаданных (как объекта

системы 1С:Предприятие), а к измерению (как объекту) регистра нужно обращаться в контексте регистра (как объекта).

Прочитать список идентификаторов измерений у заданного регистра можно следующим способом (листинг 4.51).

**Листинг 4.51. Чтение списка идентификаторов измерений у заданного регистра**

```
function Get_MeasureOfRegs_to_List(handle1C:OleVariant;
                          idReg:string;List:TStrings): integer;
// Получение списка идентификаторов измерений регистра:
// handle1C — ссылка на 1С,
// idReg    — идентификатор регистра,
// List     — здесь формируется список идентификаторов
//            измерений регистра.
//            Переменная List должна быть создана в вызывающем модуле.
// Возвращаемое значение: количество прочитанных измерений.
Var
  id: string;
  OleBuf: OleVariant;
  Yes: byte;
  i,
  Count: integer;
begin
  Result:=0;
  if List<>NIL then begin
     List.Clear;
     idReg:=Trim(idReg);
     if length(idReg)>0 then begin
        if not VarIsEmpty(handle1C) then begin
          // Определяем количество измерений в регистре
          Count:=handle1C.Метаданные.Регистр(''+idReg).Измерение*1;
          // Организуем цикл по количеству измерений
          for i:=1 to Count do
           begin
            // Получаем ссылку на соответствующий объект метаданных
            OleBuf:=handle1C.Метаданные.Регистр(''+idReg).Измерение(i*1);
            // Считываем идентификатор измерения
            id:=Trim(OleBuf.Идентификатор);
```

```
            // Добавляем идентификатор в список
            List.ADD(id);
          end;
        end;
      end;
      Result:=List.Count;
  end;
end;
```

Как видите, и здесь принцип обращения к свойствам и методам объектов `Измерение` такой же, как и к другим объектам метаданных.

Точно таким же способом считывается информация по ресурсам и реквизитам регистров. Только слово "Измерение" заменяется словом "Ресурс" или "Реквизит" соответственно.

***Примечание***

Полный текст функций, решающих эти задачи, приведен в файле \sp_Lib\sp_Lib_Delphi.PAS на сопроводительном компакт-диске книги.

На рис. 4.160 представлен результат использования этих функциий.

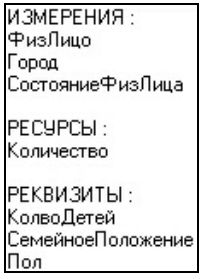



**Рис. 4.160.** Список идентификаторов измерений, ресурсов и реквизитов в регистре `УчетИзмененийСостоянияФизЛиц`

Теперь попробуем прочитать информационное содержание движений по регистрам.

Вернемся к нашему Delphi-проекту sp_1Cv7_Viewer_via_OLE.DPR.

***Примечание***

Исходный текст проекта расположен на сопроводительном компакт-диске книги в папке \Part_1\Chapter_04\sp_1Cv7_Viewer_via_OLE.PAS\06.

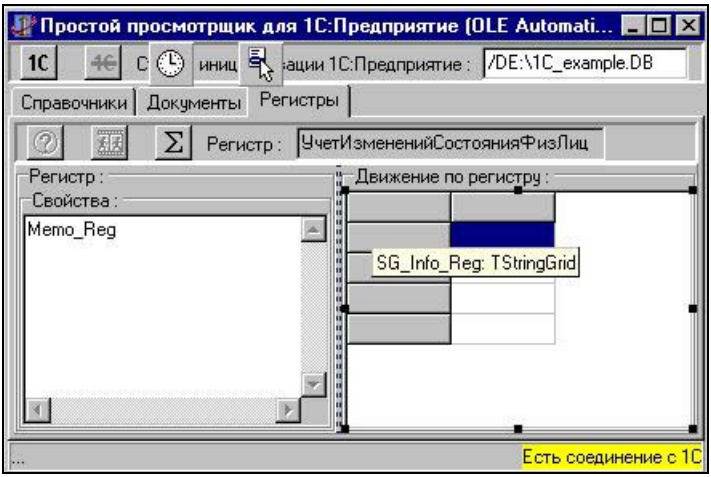



**Рис. 4.161.** Доработка формы Delphi-проекта для работы с регистрами систмы 1С:Предприятие

Создадим новую вкладку **Регистры** и расположим на ней соответствующие компоненты (рис. 4.161).

- Кнопка с вопросительным знаком (`SB_Reg_Select`) предназначена для вызова формы (`Dlg_Sel_Reg`) выбора регистра из списка регистров (см. рис. 4.158). Текст функции, создающей список регистров, рассмотрен ранее. А метод `Execute` формы (где вызывается данная функция и производится выбор из списка реквизитов) тривиален, поэтому его исходный текст здесь не приводится. После того как регистр был выбран из списка, формируется (в свойстве компонента `Memo_Reg.Lines`) список идентификаторов измерений, ресурсов и реквизитов (рис. 4.160). Текст обработчика события "нажатие этой кнопки" представлен в листинге 4.52).
- Кнопка с "бегущими человечками" (`SB_Reg_Load_Move`) предназначена для считывания движений по выбранному регистру. Этот вопрос требует более детального рассмотрения (см. далее).
- Кнопка с символом суммы (`SB_Reg_Load_Sum`) предназначена для считывания *остатков* по выбранному регистру. Этот вопрос также требует разъяснений и будет рассмотрен далее.

**Листинг 4.52. Обработчик события "нажатие кнопки `SB_Reg_Select`"**

```
procedure TForm1.SB_Reg_SelectClick(Sender: TObject);
Var
  List: TStrings;
```

```
  iCol,iRow: integer;
  Sx: string;
begin
  // Вызываем форму выбора регистра из списка регистров
  if Dlg_Sel_Reg.Execute then begin
     Memo_Reg.Lines.Clear;
     SG_Info_Reg_Clear;
     // Идентификатор выбранного регистра
     Edit_RegID.Text:=Dlg_Sel_Reg.idReg;
     List:= TStringList.Create;
     TRY
       //---------------------------------
       // Считываем список идентификаторов
       // измерений регистра
       Memo_Reg.Lines.Add('ИЗМЕРЕНИЯ:');
       Get_MeasureOfRegs_to_List(ole_1C,Edit_RegID.Text,List);
       Memo_Reg.Lines.AddStrings(List);
       for iCol:=0 to (List.Count-1) do
        begin
          SG_Info_Reg.ColCount:=SG_Info_Reg.ColCount+1;
          SG_Info_Reg.Cells[SG_Info_Reg.ColCount-1,0]:=List[iCol];
       end;
       //---------------------------------
       //---------------------------------
       // Считываем список идентификаторов
       // ресурсов регистра
       Memo_Reg.Lines.Add('');
       Memo_Reg.Lines.Add('РЕСУРСЫ:');
       Get_ResOfRegs_to_List(ole_1C,Edit_RegID.Text,List);
       Memo_Reg.Lines.AddStrings(List);
       for iCol:=0 to (List.Count-1) do
        begin
          SG_Info_Reg.ColCount:=SG_Info_Reg.ColCount+1;
          SG_Info_Reg.Cells[SG_Info_Reg.ColCount-1,0]:=List[iCol];
       end;
       //---------------------------------
       //---------------------------------
       // Считываем список идентификаторов
       // реквизитов регистра
```

```
      Memo_Reg.Lines.Add('');
      Memo_Reg.Lines.Add('РЕКВИЗИТЫ:');
      Get_AttrOfRegs_to_List(ole_1C,Edit_RegID.Text,List);
      Memo_Reg.Lines.AddStrings(List);
      for iCol:=0 to (List.Count-1) do
       begin
         SG_Info_Reg.ColCount:=SG_Info_Reg.ColCount+1;
         SG_Info_Reg.Cells[SG_Info_Reg.ColCount-1,0]:=List[iCol];
      end;
      //--------------------------------
      SB_Reg_Load_Move.Enabled:=TRUE;
    FINALLY
      if List<>NIL then begin
         List.Clear;
         List.Free;
         List:=NIL;
      end;
    END;
  end;
end;
```

Поясним текст обработчика.

Сначала вызывается метод `Execute` формы `Dlg_Sel_Reg`, где формируется список идентификаторов всех регистров и выбирается один из регистров.

Далее считывается идентификатор выбранного регистра:

```
Edit_RegID.Text:=Dlg_Sel_Reg.idReg;
```

Затем поочередно формируются списки:

- идентификаторов измерений (`Get_MeasureOfRegs_to_List`);
- идентификаторов ресурсов (`Get_ResOfRegs_to_List`);
- идентификаторов реквизитов (`Get_AttrOfRegs_to_List`).

В процессе считывания информации из регистров часто требуется преобразовать значение агрегатного типа, возвращаемое методом `ПолучитьАтрибут` (см. далее), в "привычные" значения типа `String`, `Real`, `Integer` и т. д. Для этого нужно, прежде всего, определить тип и вид возвращаемого значения. Но чтобы определить тип и вид возвращаемого значения, нужно знать, что именно (измерение, ресурс или реквизит) идентифицирует переменная, передаваемая в качестве параметра методу `ПолучитьАтрибут`.

В листинге 4.53 приведен пример проверки существования в заданном регистре измерения с заданным идентификатором.

**Листинг 4.53. Проверка существования заданного измерения в заданном регистре**

```
function YesMeasureInReg_1C(handle1C: OleVariant; idReg,
                            idMeasure: string): boolean;
// Проверка: существует ли заданное измерение у заданного регистра:
// handle1C  — ссылка на 1С,
// idReg     — идентификатор регистра,
// idMeasure — тестируемый идентификатор измерения.
Var
  oleBuf: OleVariant;
begin
 Result:=FALSE;
 if not VarIsEmpty(handle1C) then begin
    idMeasure:=Trim(idMeasure);
    idReg:=Trim(idReg);
    if (length(idMeasure)>0) and (length(idReg)>0) then begin
       // Получаем ссылку на заданное измерение
OleBuf:=handle1C.Метаданные.Регистр(''+idReg).Измерение(''+idMeasure);
       // Проверяем корректность ссылки
       if oleBuf.Выбран>0 then begin
          Result:=TRUE; // Такое измерение существует
       end;
    end;
 end;
end;
```

Текст функции `YesMeasureInReg_1C` достаточно прост и не требует пояснений.

Точно так же проверяется существование ресурса и реквизита. Единственное различие в том, что вместо слова "Измерение" используется слово "Ресурс" или "Реквизит" соответственно (тексты этих функций — `YesResInReg_1C` и `YesAttrInReg_1C` — здесь не приводятся).

Теперь (как и в случае с документами, см. листинг 4.35) объединим все три функции в одну (`YesRMAInReg_1C`). Текст этой функции приведен в листинге 4.54).

**Листинг 4.54. Проверка присутствия заданного идентификатора среди измерений, ресурсов или реквизитов**

```
function YesRMAInReg_1C(handle1C: OleVariant;idReg,
                        idRMA: string): integer;
// Проверка: присутствует ли идентификатор idRMA
// среди измерений, ресурсов или реквизитов заданного регистра:
// handle1C — ссылка на 1С,
// idReg — идентификатор регистра,
// idRMA — тестируемый идентификатор.
// Возвращаемое значение:
//   0 — заданный идентификатор не корректен для данного регистра,
//   1 — идентификатор присутствует среди измерений регистра,
//   2 — идентификатор присутствует среди ресурсов регистра,
//   3 — идентификатор присутствует среди реквизитов регистра.
begin
  Result:=0;
  // Проверка присутствия идентификатора среди измерений
  if YesMeasureInReg_1C(handle1C,idReg,idRMA) then begin
    Result:=1;
  end
  else begin
  // Проверка присутствия идентификатора среди ресурсов
    if YesResInReg_1C(handle1C,idReg,idRMA) then begin
      Result:=2;
    end
    else begin
    // Проверка присутствия идентификатора среди реквизитов
      if YesAttrInReg_1C(handle1C,idReg,idRMA) then begin
        Result:=3;
      end;
    end;
  end;
end;
```

Текст этой функции прост и не требует пояснений.

***Примечание***

Полный текст этих функций приведен в файле \sp_Lib\sp_Lib_Delphi.PAS на сопроводительном компакт-диске книги.

Теперь более подробно рассмотрим процесс считывания информации о движении по выбранному регистру.

Как и с другими объектами агрегатного типа (рассмотренными ранее), сначала нужно получить ссылку на объект типа `Регистр` заданного вида:

```
ole_1C_reg:=ole_1C.CreateObject('Регистр.Здесь_Идентификатор_Регистра');
```

Затем нужно вызвать метод `ВыбратьДвижения` регистра, чтобы открыть выборку по регистру:

```
// Выбрать все записи о движении
if ole_1C_reg.ВыбратьДвижения>0 then begin
```

или

```
// Выбрать записи о движении за период
if ole_1C_reg.ВыбратьДвижения(''+sDate1,''+sDate2)>0 then begin
```

Далее следует организовать цикл для считывания записей движения по регистру из выборки, используя метод `ПолучитьДвижение`:

```
while ole_1C_reg.ПолучитьДвижение>0 do
```

Внутри цикла, используя соответствующие методы и свойства объекта, надо прочитать информацию о движении из текущей записи.

Например, так можно определить, приход это или расход:

```
if ole_1C_reg.Приход*1>0 then begin
  SG_Info_Reg.Cells[0,SG_Info_Reg.RowCount-1]:='+';
end
else begin
  SG_Info_Reg.Cells[0,SG_Info_Reg.RowCount-1]:='-';
end;
```

А так — прочитать номер строки документа (создавшего эту запись движения):

```
SG_Info_Reg.Cells[4,SG_Info_Reg.RowCount-1]:=
                                        Trim(ole_1C_reg.НомерСтроки);
```

Чтобы прочитать информацию о документе, создавшем эту запись, нужно получить ссылку на этот документ. Для этого используется свойство `ТекущийДокумент`:

```
oleTekDoc:= ole_1C_reg.ТекущийДокумент;
```

После этого можно с помощью свойств и методов документа получить дополнительную информацию. Например, прочитать номер, дату и время документа:

```
SG_Info_Reg.Cells[1,SG_Info_Reg.RowCount-1]:=
                                      'N '+Trim(oleTekDoc.НомерДок);
SG_Info_Reg.Cells[2,SG_Info_Reg.RowCount-1]:=Trim(oleTekDoc.ДатаДок);
SG_Info_Reg.Cells[3,SG_Info_Reg.RowCount-1]:=
                                      Trim(oleTekDoc.ПолучитьВремя);
```

Получить значения измерений, ресурсов и реквизитов из текущей записи можно так:

```
OleBuf:= ole_1C_reg.ПолучитьАтрибут(''+idRMA);
```

Здесь **idRMA** — идентификатор измерения, ресурса или реквизита.

> ***Примечание***
>
> Метод ПолучитьАтрибут возвращает не само значение (в виде строки или числа), а ссылку на объект (типа OleVariant). Чтобы можно было оперировать "привычными" в Delphi значениями, нужно конвертировать полученное значение в тип String, Real или Integer. Этот вопрос детально рассматривался ранее.

Применительно к регистрам можно отметить следующие отличия:

1. Сначала нужно определить, что именно идентифицирует переменная idRMA (измерение, ресурс или реквизит). Для этого можно использовать функцию, текст которой приведен в листинге 4.54):

   ```
   // Определим, что именно идентифицирует переменная idRMA:
   // измерение, ресурс или реквизит?
   Nr:=YesRMAInReg_1C(handle1C,Trim(ole_1C_reg.Вид),idRMA);
   ```

2. Получить ссылку на этот регистр (как объект метаданных):

   ```
   oleBuf1:=handle1C.Метаданные.Регистр(''+Trim(ole_1C_reg.Вид));
   ```

3. Определить тип и вид полученного значения:

   ```
   if Nr = 1 then begin
      sType:=Trim(oleBuf1.Измерение(''+idRMA).Тип);
      sKind:=Trim(oleBuf1.Измерение(''+idRMA).Вид);
   end;
   if Nr = 2 then begin
      sType:=Trim(oleBuf1.Ресурс(''+idRMA).Тип);
      sKind:=Trim(oleBuf1.Ресурс(''+idRMA).Вид);
   end;
   if Nr = 3 then begin
      sType:=Trim(oleBuf1.Реквизит(''+idRMA).Тип);
      sKind:=Trim(oleBuf1.Реквизит(''+idRMA).Вид);
   end;
   ```

4. Наконец, в зависимости от значений переменных sType и sKind нужно привести прочитанное значение **oleBuf** к типу string, integer, real и т. п. Этот момент был рассмотрен в предыдущих разделах.

***Примечание***

Текст функции `Get_ValueRMA_from_Reg_str`, решающей эту задачу, приведен в файле \sp_Lib\sp_Lib_Delphi.PAS на сопроводительном компакт-диске книги.

Текст обработчика события "нажатие кнопки **Прочитать движения по регистру** (`SB_Reg_Load_Move`)" приведен в листинге 4.55.

**Листинг 4.55. Обработчик события "нажатие кнопки `SB_Reg_Load_Move`"**

```
procedure TForm1.SB_Reg_Load_MoveClick(Sender: TObject);
Var
  oleBuf,
  oleTekDoc,  // Ссылка на текущий документ
  ole_1C_reg: OleVariant; // Ссылка на регистр
  iRow,iCol: integer;
  idRMA,  // Идентификатор измерения,ресурса или реквизита
  Sx: string;
begin
 if not VarIsEmpty(ole_1C) then begin
   // Получаем ссылку на выбранный регистр
   ole_1C_reg:=ole_1C.CreateObject('Регистр.'+Trim(Edit_RegID.Text));
   if not VarIsEmpty(ole_1C_reg) then begin
     // Открываем выборку движения по регистру
// if ole_1C_reg.ВыбратьДвижения('01.01.2003','03.01.2003')>0 then begin
     if ole_1C_reg.ВыбратьДвижения>0 then begin
       SB_Reg_Load_Move.Enabled:=FALSE;
  GroupBox_Reg.Caption:='Движение по регистру: ' + Trim(Edit_RegID.Text);
       // Организуем цикл получения записей из выборки
       iRow:=0;
       while ole_1C_reg.ПолучитьДвижение>0 do
        begin
        iRow:=iRow+1;
        if iRow>1 then begin
          SG_Info_Reg.RowCount:=SG_Info_Reg.RowCount+1;
        end;
//-------------------------
// Считываем тип движения (приход или расход)
        if ole_1C_reg.Приход*1>0 then begin
          SG_Info_Reg.Cells[0,SG_Info_Reg.RowCount-1]:='+';
```

```
        end
        else begin
          SG_Info_Reg.Cells[0,SG_Info_Reg.RowCount-1]:='-';
        end;
//--------------------------
//Получаем ссылку на текущий документ
        oleTekDoc:= ole_1C_reg.ТекущийДокумент;
//--------------------------
// Считываем информацию о документе, создавшем эту запись
 SG_Info_Reg.Cells[1,SG_Info_Reg.RowCount-1]:='N '
                                          + Trim(oleTekDoc.НомерДок)
                                          + ' '
                                          + Trim(oleTekDoc.Вид);
 SG_Info_Reg.Cells[2,SG_Info_Reg.RowCount-1]:= Trim(oleTekDoc.ДатаДок);
 SG_Info_Reg.Cells[3,SG_Info_Reg.RowCount-1]:=
                                        Trim(oleTekDoc.ПолучитьВремя);
//--------------------------
        SG_Info_Reg.Cells[4,SG_Info_Reg.RowCount-1]:=
                                         Trim(ole_1C_reg.НомерСтроки);
//--------------------------
// Считываем значения всех измерений, ресурсов и реквизитов
// текущей записи
        for iCol:=5 to (SG_Info_Reg.ColCount-1) do
         begin
       // Считываем идентификатор из заголовка колонки таблицы
         idRMA:=SG_Info_Reg.Cells[iCol,0];
       // Считываем значение для idRMA
         Sx:=Get_ValueRMA_from_Reg_str(ole_1C,ole_1C_reg,idRMA);
       // Записываем значение в ту же колонку, где idRMA
       // но в строку, соответствующую текущей записи движения регистра
         SG_Info_Reg.Cells[iCol,SG_Info_Reg.RowCount-1]:=Sx;
        end;
//--------------------------
        Application.ProcessMessages;
       end;
     end;
   end;
 end;
end;
```

Приведенный текст обработчика события содержит исчерпывающий комментарий. Единственное, что следует отметить: считывание значения измерения (ресурса или реквизита)

```
OleBuf:= ole_1C_reg.ПолучитьАтрибут(''+idRMA);
```

производится внутри функции `Get_ValueRMA_from_Reg_str`.

На рис. 4.162 приведен результат считывания движения по регистру `УчетИзменениеСостоянияФизЛиц` из "нашей" (учебной) конфигурации.

Движение по регистру : УчетИзменениеСостоянияФизЛиц

| +/- | ТекущийДокумен | Дата | Время | Нстр | ФизЛицо | Город | СостояниеФизЛ | Колич | КолвоДе | СемейноеПолос | Пол |
|---|---|---|---|---|---|---|---|---|---|---|---|
| - | N 1 ИзменениеСо | 28.06.2006 | 23:38:42 | 1 | Аксакалов А | Киев | СотрудникНеоси | 1 | 3 | Разведен | Мужс |
| + | N 1 ИзменениеСо | 28.06.2006 | 23:38:42 | 1 | Аксакалов А | Киев | СотрудникОсно | 1 | 3 | Разведен | Мужс |
| - | N 1 ИзменениеСо | 28.06.2006 | 23:38:42 | 2 | Федоров Ф | Киев | ВременноНеРаб | 1 | 2 | Разошелся | Мужс |
| + | N 1 ИзменениеСо | 28.06.2006 | 23:38:42 | 2 | Федоров Ф | Киев | СотрудникНеоси | 1 | 2 | Разошелся | Мужс |
| - | N 1 ИзменениеСо | 28.06.2006 | 23:38:42 | 3 | Криогенова | Гомел | СотрудникОсно | 1 | 1 | НикогдаНеСос | Жен |
| + | N 1 ИзменениеСо | 28.06.2006 | 23:38:42 | 3 | Криогенова | Гомел | ВременноНеРаб | 1 | 1 | НикогдаНеСос | Жен |

**Рис. 4.162.** Движение по регистру `УчетИзменениеСостоянияФизЛиц`

На рис. 4.163 приведен результат считывания движения по регистру `ПартииНаличие` из реальной конфигурации 1С:Предприятие.

Движение по регистру : ПартииНаличие

| +/- | ТекущийДокумент | Дата | Время | Нстр | Фирма | МОЛ | Номенклатура | СтатусПартии | Колич | СуммаРуб | СуммаБезНДС | КодОперации |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| + | N 0000000113 ПоступлениеТМЦ | 01.01 | 06:00:3 | 1 | ОАО "( | Габду | Пиловочник хв | М_Сырье | 64.58 | 87183 | 72652.5 | ЗакупкаИнвес |
| + | N 0000000002 ПоступлениеДопРас | 01.01 | 06:00:5 | 0 | ОАО "( | Габду | Пиловочник хв | М_Сырье | 0 | 39414.88 | 32845.73 | Дополнительн |
| + | N 0000000121 ПоступлениеТМЦ | 01.01 | 06:01:1 | 1 | ОАО "( | Габду | Пиломатериал | М_Сырье | 72.86 | 207651 | 173042.5 | ЗакупкаИнвес |
| - | N 0000000075 ПеремещениеТМЦ | 01.01 | 06:01:3 | 1 | ОАО "( | Бурми | Сталь листова | М_Сырье | 99.94 | 2412030.41 | 2014110.01 | Перемещение |
| + | N 0000000075 ПеремещениеТМЦ | 01.01 | 06:01:3 | 1 | ОАО "( | Габду | Сталь листова | М_Сырье | 99.94 | 2412030.41 | 2014110.01 | Перемещение |
| - | N 0000000076 ПеремещениеТМЦ | 01.01 | 06:01:4 | 1 | ОАО "( | Бурми | Утяжелитель У | М_Сырье | 162 | 232891.2 | 194076 | Перемещение |
| + | N 0000000076 ПеремещениеТМЦ | 01.01 | 06:01:4 | 1 | ОАО "( | Габду | Утяжелитель У | М_Сырье | 162 | 232891.2 | 194076 | Перемещение |
| - | N 0000000709 ПеремещениеТМЦ | 01.01 | 06:01:5 | 1 | ОАО "( | Бариг | Рукавицы ватн | М_Спецодежд | 20 | 898.26 | 745.69 | Перемещение |
| + | N 0000000709 ПеремещениеТМЦ | 01.01 | 06:01:5 | 1 | ОАО "( | Шаба | Рукавицы ватн | М_Спецодежд | 20 | 898.26 | 745.69 | ВводИнвентар |
| - | N 0000000826 ПеремещениеТМЦ | 01.01 | 06:02:0 | 1 | ОАО "( | Лазар | Кабель экрани | М_Сырье | 120 | 34214.4 | 28512 | Перемещение |
| - | N 0000000826 ПеремещениеТМЦ | 01.01 | 06:02:0 | 2 | ОАО "( | Лазар | Ввод кабельны | М_Сырье | 2 | 1298.88 | 1082.4 | Перемещение |

**Рис. 4.163.** Движение по регистру `ПартииНаличие` из реальной конфигурации 1С:Предприятие

Считывание информации с использованием механизма OLE Automation — *весьма медленный процесс.* Крайне рекомендуется при программировании учитывать это и предоставлять пользователю возможность считывать информацию в ограниченном объеме (например, за некоторый период или только то, что привязано к какому-то документу).

В данном случае (см. рис. 4.163) из регистра считывалась не вся информация о движении:

```
if ole_1C_reg.ВыбратьДвижения>0 then begin
```

а только за небольшой временной период:

```
if ole_1C_reg.ВыбратьДвижения('01.01.2003','03.01.2003')>0 then begin
```

Иначе пришлось бы ожидать окончания процесса несколько часов…

Как уже говорилось, регистр остатков предназначен для ведения остатков ресурсов. То есть из регистра можно получить информацию не только о движении, но и об остатках ресурсов как в целом, так и в разрезе измерений.

Здесь следует отметить, что программирование считывания остатков (посредством механизма OLE Automation), в отличие от считывания движения, существенно затруднено по той причине, что обращение к методам `Остаток` и `СводныйОстаток` регистра описанным в документации способом не приводит к ожидаемому результату. Достаточно корректным и оптимальным выходом из этой ситуации может быть использование *комбинированного* способа считывания остатков из регистра. Суть его сводится к тому, что в глобальном модуле конфигурации штатными средствами системы 1С:Предприятие создаются соответствующие функции (с ключевым словом "Экспорт"), реализующие алгоритмы получения остатков по регистру. А во внешнем приложении эти функции вызываются (посредством OLE Automtion) с помощью метода `EvalExpr`. Таким образом, функция глобального модуля конфигурации корректно производит всю обработку информации. А внешнее приложение получает результат, возвращаемый этой функцией.

Рассмотрим этот способ на конкретном примере. Для этого вновь внесем изменения в конфигурацию, введя:

- два новых перечисления (рис. 4.164);
- новый регистр остатков `УчетВыплатФизЛицам` (рис. 4.165);
- новый документ `ВыплатыФизЛицам` (рис. 4.166).

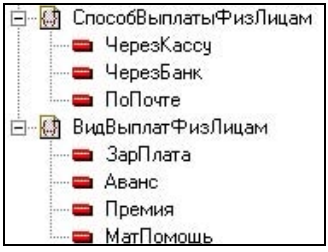



**Рис. 4.164.** Новые перечисления, "обслуживающие" регистр `УчетВыплатФизЛицам`

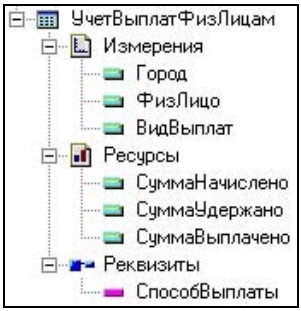



**Рис. 4.165.** Новый регистр `УчетВыплатФизЛицам`

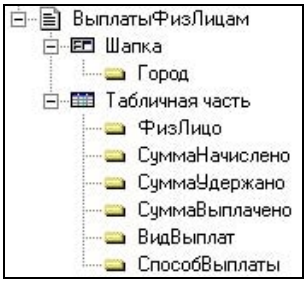



**Рис. 4.166.** Новый документ,
"обслуживающий" регистр `УчетВыплатФизЛицам`

В документе `ВыплатыФизЛицам` установим признак оперативного учета (флажок **Оперативный учет** на рис. 4.153). Создадим форму документа, как уже делали это (см. рис. 4.108 и 4.109). И изменим соответствующим образом текст процедуры проведения документа (листинг 4.56).

**Листинг 4.56. Процедура `ОбработкаПроведения` документов вида `ВыплатыФизЛицам`**

```
// Обработка проведения документов вида "ВыплатыФизЛицам"
Процедура ОбработкаПроведения()
  Если Город.Выбран()>0 Тогда
    ВыбратьСтроки();
    Пока ПолучитьСтроку() = 1 Цикл
      ПривязыватьСтроку(НомерСтроки);
      Если ФизЛицо.Выбран()>0 тогда
```

```
        //------------------
        // Приход по старому состоянию физ.лица
        // Измерения
        Регистр.УчетВыплатФизЛицам.ФизЛицо = ФизЛицо;
        Регистр.УчетВыплатФизЛицам.Город = Город;
        Регистр.УчетВыплатФизЛицам.ВидВыплат = ВидВыплат;
        // Реквизиты
        Регистр.УчетВыплатФизЛицам.СпособВыплаты = СпособВыплаты;
        // Ресурсы
        Регистр.УчетВыплатФизЛицам.СуммаНачислено = СуммаНачислено;
        Регистр.УчетВыплатФизЛицам.СуммаУдержано  = СуммаУдержано;
        Регистр.УчетВыплатФизЛицам.СуммаВыплачено = СуммаВыплачено;

        // Движение: Приход
        Регистр.УчетВыплатФизЛицам.ДвижениеПриходВыполнить();
        //------------------

      Иначе
        Сообщить("ФизЛицо не указано");
      КонецЕсли;
    КонецЦикла;
  Иначе
    Сообщить("Город не указан");
  КонецЕсли;
КонецПроцедуры
```

Как видите, в рамках этого документа будут создаваться только записи типа `Приход` по регистру `УчетВыплатФизЛицам`. Это связано с тем, что для нас (в данный момент) неважно, насколько правильным будет такой подход с точки зрения бухучета. Для нас важна корректность с точки зрения темы этой книги (програмимирование информационного обмена с системой 1С:Предприятие). Поэтому достаточно будет только "прихода".

Откроем глобальный модуль конфигурации (рис. 4.167).

И в глобальном модуле напишем функцию, которая выбирает остаток по регистру `УчетВыплатФизЛицам` с фильтром по физическому лицу (листинг 4.57).

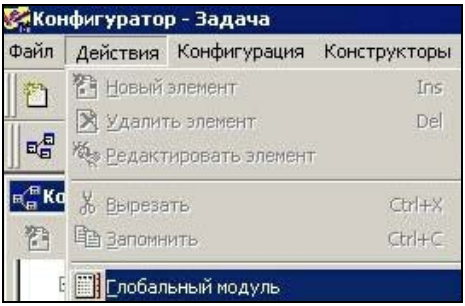



**Рис. 4.167.** Чтобы открыть глобальный модуль, нужно сначала открыть конфигурацию, а затем выбрать соответствующий пункт меню

**Листинг 4.57. Функция глобального модуля, возвращающая остаток ресурса СуммаВыплачено регистра УчетВыплатФизЛицам**

```
// Получение суммы выплат по физ.лицу из регистра "УчетВыплатФизЛицам".
// Функция предназначена для вызова из внешних приложений (и не только)
Функция гл9_ПолучитьСуммуВыплат_для_ФизЛица(КодФизЛица) Экспорт
  Сумма = 0; // Переменная, значение которой вернет эта функция.
  // Поскольку в функцию передается код физ.лица,
  // а для фильтрации нужен элемент справочника,
  // мы должны найти этот элемент в справочнике "ФизЛица".
  СпрФизЛиц = СоздатьОбъект("Справочник.ФизЛица");
  Если СпрФизЛиц.НайтиПоКоду(""+СокрЛП(КодФизЛица),0)>0 Тогда
     // Получаем ссылку на свойство "ТекущийЭлемент"
     // справочника "ФизЛица"
     ФизЛицо = СпрФизЛиц.ТекущийЭлемент();
     // Получаем ссылку на регистр
     Рег = Регистр.УчетВыплатФизЛицам;
     // Указываем идентификатор ресурса, по которому
     // будем получать остаток
     idRes = "СуммаВыплачено";
     // Получаем остаток по ресурсу "СуммаВыплачено"
     // в разрезе измерения "ФизЛицо"
     Сумма = Рег.СводныйОстаток(,ФизЛицо,,""+idRes);
     //--------------------------
     // Эти строки — только для отладки и иллюстрации.
```

```
    // Их нужно закомментировать.
    ТекстПредпр = "Сумма выплат для "
                + СокрЛП(ФизЛицо.Наименование)
                + "="
                + СокрЛП(Сумма);
    Предупреждение(ТекстПредпр);
    //-------------------------
  Иначе
    Предупреждение("ФизЛицо с кодом="+СокрЛП(КодФизЛица)+" не найдено");
  КонецЕсли;
  // Возвращем значение остатка
  Возврат Сумма;
КонецФункции
```

Здесь требуются некотрые пояснения. Дело в том, что если пользоваться только штатными средствами 1С, то вместо кода `ФизЛица` можно было бы передать значение агрегатного типа (ссылка на элемент справочника `ФизЛица`). Но в случае доступа из внешних приложений такая возможность отсутствует. Поэтому в функцию передается код элемента справочника.

Здесь, в строке

```
Сумма = Рег.СводныйОстаток(,ФизЛицо,,""+idRes);
```

перед значением **ФизЛицо** оставлено "пустое место" перед запятой. Это не ошибка. Дело в том, что когда мы создавали регистр `УчетВыплатФизЛицам`, первым измерением был `Город` (см. рис. 4.165). В данном случае нас интересует остаток ресурса в разрезе измерения `ФизЛицо` (которое следует вторым в списке). Синтаксис встроенного языка программирования системы 1С:Предприятие допускает такое обращение.

> ***Примечание***
>
> Более подробно назначение используемых в функции `гл9_ПолучитьСуммуВыплат_для_ФизЛица` свойств и методов описано в специальной литературе по системе 1С:Предприятие.

Аналогично нужно написать функции, возвращающие значения остатков по другим измерениям регистра `УчетВыплатФизЛицам` (здесь тексты этих функций не приводятся).

Сохраним изменения в конфигурации, запустим программу 1С:Предприятие и затем создадим и проведем три документа вида `ВыплатыФизЛицам` (рис. 4.168—4.170).

НомерДок: 1 ДатаДок: 14.07.06

Город: Волгоград Комментарий:

| N | ФизЛицо | СуммаНачислено | СуммаУдержано | СуммаВыплач... | ВидВыплат | СпособВыплаты |
|---|---|---|---|---|---|---|
| 1 | Аксакалов Аксака | 1000 | 10 | 990 | ЗарПлата | ПоПочте |
| 2 | Аксакалов Аксака | 30 | 1 | 29 | МатПомощь | ПоПочте |
| 3 | Сидоров Сидор Си | 300 | 3 | 297 | Премия | ЧерезБанк |

**Рис. 4.168.** Первый новый документ вида `ВыплатыФизЛицам`

НомерДок: 2 ДатаДок: 15.07.06

Город: Одесса Комментарий:

| N | ФизЛицо | СуммаНачислено | СуммаУдержано | СуммаВыплачено | ВидВыплат | СпособВыплаты |
|---|---|---|---|---|---|---|
| 1 | Иванов Иван Иванов | 100 | 1 | 99 | Аванс | ПоПочте |
| 2 | Криогенова Фекла М | 200 | 2 | 198 | МатПомощь | ПоПочте |

**Рис. 4.169.** Второй новый документ вида `ВыплатыФизЛицам`

НомерДок: 3 ДатаДок: 29.06.06

Город: Барановичи Комментарий:

| N | ФизЛицо | СуммаНачис... | СуммаУдерж... | СуммаВыпла... | ВидВыплат | СпособВыплаты |
|---|---|---|---|---|---|---|
| 1 | Петров Петр Петр | 1000 | 10 | 990 | Аванс | ЧерезКассу |
| 2 | Криогенова Фекл | 300 | 3 | 297 | Премия | ПоПочте |

**Рис. 4.170.** Третий новый документ вида `ВыплатыФизЛицам`

Движение по регистру : УчетВыплатФизЛицам

| +/- | ТекущийДокум | Дата | Время | Nст | Город | ФизЛиц | ВидВыплат | СуммаНачис | СуммаУде | СуммаВып | СпособВыплат |
|---|---|---|---|---|---|---|---|---|---|---|---|
| + | N 3 ВыплатыФи | 29.06.2006 | 12:26: | 1 | Баранс | Петров П | Аванс | 1000 | 10 | 990 | ЧерезКассу |
| + | N 3 ВыплатыФи | 29.06.2006 | 12:26: | 2 | Баранс | Криогенс | Премия | 300 | 3 | 297 | ПоПочте |
| + | N 1 ВыплатыФи | 14.07.2006 | 12:00: | 1 | Волгог | Аксакалс | ЗарПлата | 1000 | 10 | 990 | ПоПочте |
| + | N 1 ВыплатыФи | 14.07.2006 | 12:00: | 2 | Волгог | Аксакалс | МатПомощь | 30 | 1 | 29 | ПоПочте |
| + | N 1 ВыплатыФи | 14.07.2006 | 12:00: | 3 | Волгог | Сидоров | Премия | 300 | 3 | 297 | ЧерезБанк |
| + | N 2 ВыплатыФи | 15.07.2006 | 11:53: | 1 | Одесс | Иванов И | Аванс | 100 | 1 | 99 | ПоПочте |
| + | N 2 ВыплатыФи | 15.07.2006 | 11:53: | 2 | Одесс | Криогенс | МатПомощь | 200 | 2 | 198 | ПоПочте |

**Рис. 4.171.** Движение по регистру `УчетВыплатФизЛицам`.

Закроем программу 1С:Предприятие, запустим наш проект на выполнение и прочитаем движение по регистру `УчетВыплатФизЛицам`. Результат представлен на рис. 4.171.

Здесь — ничего нового (получили то, что и должно было быть).

Теперь посмотрим, каким образом мы можем получить остатки ресурсов регистра `УчетВыплатФизЛицам` по любому физическому лицу с использованием разработанных ранее функций (`гл9_ПолучитьСуммуВыплат_для_ФизЛица` и других, аналогичных).

В листинге 4.58 приведен текст обработчика события "нажатие кнопки с символом суммы (`SB_Reg_Load_Sum`) ".

**Листинг 4.58. Обработчик события "нажатие кнопки с символом суммы (SB_Reg_Load_Sum)"**

```
procedure TForm1.SpeedButton1Click(Sender: TObject);
Var
  oleBuf: OleVariant;
  sCode,
  Sx: string;
begin
 if not VarIsEmpty(ole_1C) then begin
    sCode:='5';
    // Укажем код элемента справочника ФизЛица, для которого
    // мы желаем получить информацию
    if InputQuery('Получить сумму выплат','для сотрудника с кодом',sCode)
    then begin
       sCode:=Trim(sCode);
       if length(sCode)>0 then begin
          // Получим остаток по ресурсу "СуммаВыплачено"
          Sx:='гл9_ПолучитьСуммуВыплат_для_ФизЛица('+sCode+')';
          oleBuf:=ole_1C.EvalExpr(''+Sx);
          ShowMessage('Выплаты для ФизЛица с кодом '+sCode+'='
                                                 + Trim(oleBuf));
          // Получим остаток по ресурсу "СуммаНачислено"
          Sx:='гл9_ПолучитьСуммуНачислений_для_ФизЛица('+sCode+')';
          oleBuf:=ole_1C.EvalExpr(''+Sx);
          ShowMessage('Начисления для ФизЛица с кодом '+sCode+'='
                                                    +Trim(oleBuf));
          // Получим остаток по ресурсу "СуммаУдержано"
          Sx:='гл9_ПолучитьСуммуУдержаний_для_ФизЛица('+sCode+')';
          oleBuf:=ole_1C.EvalExpr(''+Sx);
          ShowMessage('Удержания для ФизЛица с кодом '+sCode+'='
```

```
                                                    +Trim(oleBuf));
        end;
    end;
 end;
end;
```

Как мне кажется, комментарии здесь требуются только для функции `EvalExpr`, но ее мы подробно рассмотрим чуть позже.

Запустим проект на выполнение. Создадим соединение с 1С:Предприятие и нажмем кнопку с символом суммы. На экране появится окно запроса, предлагающее ввести код физического лица, для которого нужно получить остатки (рис. 4.172).

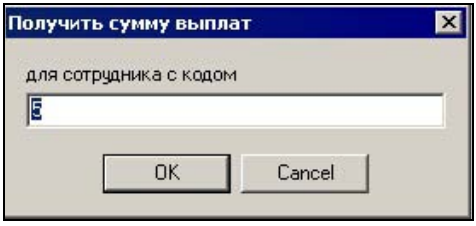



**Рис. 4.172.** Запрос на ввод кода физического лица

После ввода кода на экране появится соответствующее сообщение (рис. 4.173). Это — результат работы функции `гл9_ПолучитьСуммуВыплат_для_ФизЛица`.

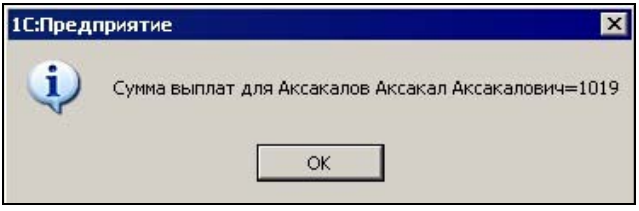



**Рис. 4.173.** Результат работы функции `гл9_ПолучитьСуммуВыплат_для_ФизЛица`

### *Примечание*

Окно программы 1С:Предприятие (как сервера OLE Automation) по умолчанию остается невидимым. Поэтому диалоговое окно, показанное на рис. 4.173, может быть скрыто другими окнами различных приложений, запущенных в данный момент на компьютере. В этом случае для "поиска" диалогового окна можно использовать сочетание клавиш <Alt>+<Tab>. На рис. 4.50 показана пиктограмма, которую нужно выбрать для того, чтобы диалоговое окно появилось поверх остальных окон.

После того как будет нажата кнопка **OK** (см. рис. 4.173), на экране появится окно с сообщением (рис. 4.174). Это уже сработал обработчик `TForm1.SpeedButton1Click`.

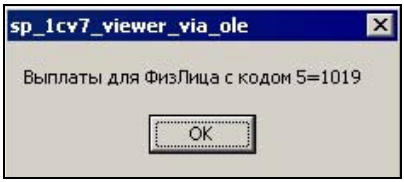



**Рис. 4.174.** Это сообщение выведено инструкцией `ShowMessage` обработчика `TForm1.SpeedButton1Click`

Таким образом, будут поочередно показаны остатки по всем ресурсам регистра `УчетВыплатФизЛицам`.

***Примечание***

Исходные тексты проекта, где реализованы вопросы считывания информации из регистров, находятся в папке \Part_1\Chapter_04\sp_1Cv7_Viewer_via_OLE.PAS\06 на сопроводительном компакт-диске книги.

***Примечание***

Представленные примеры исходных текстов — это упрощенная иллюстрация к рассматриваемым вопросам. Назначение этих примеров — дать представление о возможном подходе при решении задач информационного обмена с системой 1С:Предприятие. При работе с реальными базами 1С может потребоваться их дальнейшая доработка.

# 4.4. Доступ к информационной структуре базы данных 1С

## 4.4.1. Доступ к информационной структуре с использованием механизма OLE Automation

В процессе разработки программного обеспечения для обмена информацией с любым "чужим" программным продуктом (информационная структура которого может гибко и достаточно часто меняться) программист, как правило, стремится в максимальной степени уйти от жестких привязок (в исходных текстах) к конкретной (текущей) реализации "чужой" информационной структуры.

В данном случае, применительно к системе 1С:Предприятие, мы имеем такую же ситуацию. В процессе производственной эксплуатации информаци-

онная структура базы данных системы 1С:Предприятие может периодически и существенно изменяться.

В этом разделе будут рассмотрены различные аспекты процесса считывания информационной структуры с использованием механизма OLE Automation.

Частично, по ходу изложения, эти вопросы были рассмотрены в предыдущих разделах. Здесь мы попытаемся разработать некое более универсальное средство для считывания информационной структуры базы данных системы 1С:Предприятие.

Запустим Delphi и создадим новый проект с именем sp_LoadStructure_1C.DPR (или любым другим по желанию).

> ***Примечание***
>
> Исходные тексты этого примера есть на CD в папке \Part_1\Chapter_04\Topic_4_4\sp_LoadStructure_1C.PAS.

Функционально наш проект должен "уметь":

- обеспечивать соединение с системой 1С:Предприятие (сервер OLE Automation);
- считывать информационную структуру базы данных открытой конфигурации системы в том объеме, какой требуется пользователю;
- сохранять прочитанную структуру в текстовом файле для того, чтобы потом можно было работать с этой информацией без соединения с 1С.

На рис. 4.175 представлен вид формы проекта. Ее основные компоненты кратко описаны в табл. 4.7.

***Таблица 4.7.*** *Назначение основных компонентов в форме проекта sp_LoadStructure_1C.DPR*

| Компонент | Тип | Описание |
|---|---|---|
| `ListWhat` | `TCheckListBox` | Расположен в правой части формы, содержит список типов метаданных, информация о которых будет считываться |
| `ListStr` | `TTreeView` | Расположен в центре формы, предназначен для размещения полученной информации о структуре базы данных системы 1С:Предприятие в виде дерева |
| `SpeedButton2` | `TSpeedButton` | Кнопка **Структура**; при ее нажатии происходит соединение с программой 1С:Предприятие как сервером OLE Automation |
| `SpeedButton6` | `TSpeedButton` | Кнопка , предназначена для экспорта информационной структуры в специфичный формат (в виде INI-файла). Этот файл понадобится нам в *части II* |

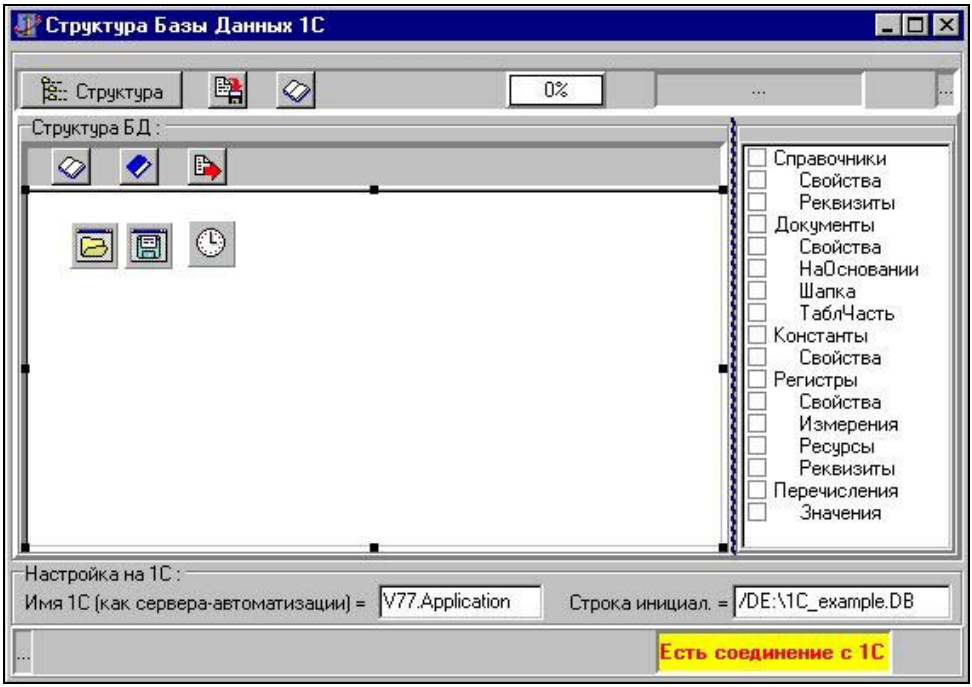



**Рис. 4.175.** Форма проекта sp_LoadStructure_1C.DPR

Вся функциональность по считыванию информационной структуры сосредоточена в обработчике события "нажатие кнопки **Структура** (`SpeedButton2`)". Текст обработчика представлен в листинге 4.59.

**Листинг 4.59. Обработчик события "нажатие кнопки *Структура* (`SpeedButton2`)"**

```
procedure TForm1.SpeedButton2Click(Sender: TObject);
// Считывание информационной структуры системы 1С:Предприятие
Var
 CountCommonRekvDoc,
 iRekv: integer;
 Sx: string;
 ListBuf: TStrings;
begin
Ole_1C_App:=UnAssigned;
 Ole_1C_App:=ConnectTo1C(Edit_1Cinit.Text); // Соединяемся с 1С
 if not VarIsEmpty(Ole_1C_App) then begin
```

```
  ListBuf:=TStringList.Create;
  TRY
   ListStr.Items.Clear;
   //---------------------------------------------
   // Считываем общую информацию о конфигурации
   ListBuf.Add('ОбщаяИнформация ['+DateTimeToStr(NOW)+']');
   Sx:=Trim(Ole_1C_App.Метаданные.Идентификатор);
   ListBuf.Add(#9+'Идентификатор='+Sx);
   Sx:=Trim(Ole_1C_App.Метаданные.ПолныйИдентификатор);
   ListBuf.Add(#9+'ПолныйИдентификатор='+Sx);
   Sx:=Trim(Ole_1C_App.Метаданные.Синоним);
   ListBuf.Add(#9+'Синоним='+Sx);
   Sx:=Trim(Ole_1C_App.Метаданные.Комментарий);
   ListBuf.Add(#9+'Комментарий='+Sx);
   Sx:=Trim(Ole_1C_App.Метаданные.Пароль);
   ListBuf.Add(#9+'Пароль='+Sx);
   Sx:=Trim(Ole_1C_App.Метаданные.АвторКороткаяИнформация);
   ListBuf.Add(#9+'АвторКороткаяИнформация='+Sx);
   Sx:=Trim(Ole_1C_App.Метаданные.Язык);
   ListBuf.Add(#9+'Язык='+Sx);
   Sx:=Trim(Ole_1C_App.Метаданные.Округление);
   ListBuf.Add(#9+'Округление='+Sx);
   Sx:=Trim(Ole_1C_App.Метаданные.НепосредственноеУдалениеОбъектов);
   ListBuf.Add(#9+'НепосредственноеУдалениеОбъектов='+Sx);
   //---------------------------------------------
   //---------------------------------------------
   // Считываем общие реквизиты документов
   Application.ProcessMessages;
   CountCommonRekvDoc:=Ole_1C_App.Метаданные.ОбщийРеквизитДокумента;
   Sx:=IntToStr(CountCommonRekvDoc);
   ListBuf.Add(#9+'ОбщийРеквизитДокумента='+Sx);
   if CountCommonRekvDoc>0 then begin
     for iRekv:=1 to CountCommonRekvDoc do
      begin
Sx:=Trim(Ole_1C_App.Метаданные.ОбщийРеквизитДокумента(iRekv).Идентификатор);
      ListBuf.Add(#9+#9+Sx);
Sx:=Trim(Ole_1C_App.Метаданные.ОбщийРеквизитДокумента(iRekv).Синоним);
      ListBuf.Add(#9+#9+#9+'Синоним='+Sx);
```

```
        Sx:=Trim(Ole_1C_App.Метаданные.ОбщийРеквизитДокумента(iRekv).Тип);
        ListBuf.Add(#9+#9+#9+'Тип='+Sx);
        Sx:=Trim(Ole_1C_App.Метаданные.ОбщийРеквизитДокумента(iRekv).Вид);
        ListBuf.Add(#9+#9+#9+'Вид='+Sx);
Sx:=Trim(Ole_1C_App.Метаданные.ОбщийРеквизитДокумента(iRekv).Длина);
        ListBuf.Add(#9+#9+#9+'Длина='+Sx);
Sx:=Trim(Ole_1C_App.Метаданные.ОбщийРеквизитДокумента(iRekv).Точность);
        ListBuf.Add(#9+#9+#9+'Точность='+Sx);
Sx:=Trim(Ole_1C_App.Метаданные.ОбщийРеквизитДокумента(iRekv).НеОтрицательный);
        ListBuf.Add(#9+#9+#9+'НеОтрицательный='+Sx);
Sx:=Trim(Ole_1C_App.Метаданные.ОбщийРеквизитДокумента(iRekv).РазделятьТриады);
        ListBuf.Add(#9+#9+#9+'РазделятьТриады='+Sx);
Sx:=Trim(Ole_1C_App.Метаданные.ОбщийРеквизитДокумента(iRekv).Комментарий)
;
        ListBuf.Add(#9+#9+#9+'Комментарий='+Sx);
        Application.ProcessMessages;
       end;
    end;
    //---------------------------------------------
    //---------------------------------------------
    // Считываем константы
    if ListWhat.Checked[8] then begin
       Application.ProcessMessages;
       GetStructConstants(ListBuf);
    end;
    //---------------------------------------------
    //---------------------------------------------
    // Считываем перечисления
    if ListWhat.Checked[15] then begin
       Application.ProcessMessages;
       GetStructEnum(ListBuf);
    end;
    //---------------------------------------------
    //---------------------------------------------
    // Считываем справочники
    if ListWhat.Checked[0] then begin
       Application.ProcessMessages;
       GetStructReference(ListBuf);
    end;
```

```
  //---------------------------------------------
  //---------------------------------------------
  // Считываем документы
  Application.ProcessMessages;
  if ListWhat.Checked[3] then begin
     Application.ProcessMessages;
     GetStructDocuments(ListBuf);
  end;
  //---------------------------------------------
  //---------------------------------------------
  // Считываем регистры
  Application.ProcessMessages;
  if ListWhat.Checked[10] then begin
     Application.ProcessMessages;
     GetStructRegisters(ListBuf);
  end;
  //---------------------------------------------
  //---------------------------------------------
  // Сохраняем информацию на диске
   ListBuf.SaveToFile(MainDir+nf_ListStr);
  //---------------------------------------------
  //---------------------------------------------
  // Загружаем информацию с диска
  ListStr.LoadFromFile(MainDir+nf_ListStr);
  //---------------------------------------------
FINALLY
  if ListBuf<>NIL then begin
     ListBuf.Clear;
     ListBuf.Free;
     ListBuf:=NIL;
  end;
END;
//------------------------
// Открываем дерево конфигурации
ListStr.FullExpand;
ListStr.Selected:=ListStr.Items[0];
ListStr.SetFocus;
```

```
   //------------------------
 end;
 Ole_1C_App:= UnAssigned; // Разрываем соединение с 1С
end;
```

Поясним текст обработчика (листинг 4.59).

Здесь следует отметить следующее. Вся информация, что будет прочитана, размещается в переменной `ListBuf` (типа `TStrings`) в текстовом формате. Это сделано для того, чтобы эту информацию можно было сохранить в текстовом файле на диске и загружать (в компонент `ListStr`) по мере необходимости, без подключения к 1С. Символы табуляции (`#9`) в строки добавлены с целью обеспечить иерархическую структуру (в виде дерева) при загрузке текстового файла в компонент `ListStr`.

Сначала считываются общие свойства конфигурации (в данном случае мы считываем далеко не все свойства из имеющихся), затем общие реквизиты документов и их свойства. Далее, в зависимости от того, что именно было отмечено флажком в списке `ListWhat`, считываем другие объекты метаданных и их свойства (функции `GetStructConstants`, `GetStructEnum`, `GetStructReference`, `GetStructDocuments`, `GetStructRegisters`).

После считывания информация сохраняется в текстовом файле:

```
ListBuf.SaveToFile(MainDir+nf_ListStr);
```

А затем информация из текстового файла загружается в компонент `ListStr`:

```
ListStr.LoadFromFile(MainDir+nf_ListStr);
```

Соединение с 1С разрывается. Вот, собственно, и все.

> ***Примечание***
>
> Способы считывания идентификаторов справочников и их реквизитов подробно рассматривались в предыдущих разделах. Поэтому здесь будут представлены только те вопросы, которых мы еще не касались.

Так же, как и ранее, всю информацию получаем, обращаясь к свойствам и методам объекта `Метаданные` системы 1С:Предприятие. В листинге 4.60 представлен фрагмент функции, считывающей основные свойства справочников заданного вида.

**Листинг 4.60. Фрагмент функции, считывающей свойства справочников заданного вида**

```
function TForm1.GetStructReference(Lines: TStrings): integer;
begin
  Result:=0;
  ...
```

```
  // Получаем ссылку на объект типа "Справочник" метаданных
  // по его порядковому номеру
  Ole0:=Ole_1C_App.Метаданные.Справочник(iRef*1);
  ...
  //-------------------------------
  // Читаем свойства справочника
  Sx:=Trim(Ole0.Идентификатор);
  Sx:=Trim(Ole0.Синоним);
  Sx:=Trim(Ole0.ОсновноеПредставление);
  Sx:=Trim(Ole0.КонтрольУникальности);
  Sx:=Trim(Ole0.СерииКодов);
  Sx:=Trim(Ole0.АвтоНумерация);
  Sx:=Trim(Ole0.КоличествоУровней);
  Sx:=Trim(Ole0.ГруппыВпереди);
  Sx:=Trim(Ole0.СпособРедактирования);
  Sx:=Trim(Ole0.ЕдинаяФормаЭлемента);
  Sx:=Trim(Ole0.Комментарий);
  //-------------------------------
  //-------------------------------
  // Читаем свойства кода элементов справочника
  Sx:=Trim(Ole0.ТипКода);
  Sx:=Trim(Ole0.ДлинаКода);
  Sx:=Trim(Ole0.СерииКодов);
  //-------------------------------
  //-------------------------------
  // Читаем свойства наименования элементов справочника
  Sx:=Trim(Ole0.ДлинаНаименования);
  //-------------------------------
  ...
end;
```

В листинге 4.61 представлен фрагмент функции, считывающей основные свойства реквизитов справочника.

**Листинг 4.61. Фрагмент функции, считывающей свойства реквизитов справочника**

```
function TForm1.GetStructReference(Lines: TStrings): integer;
begin
  Result:=0;
  ...
```

```
  // Получаем ссылку на объект типа "Реквизит" справочников
  // заданного вида по его порядковому номеру
  Ole1:=Ole_1C_App.Метаданные.Справочник(iRef*1).Реквизит(iAttr*1);
  //--------------------------------
  // Читаем свойства реквизита справочника
  Sx:=Trim(Ole1.Идентификатор);
  Sx:=Trim(Ole1.Синоним);
  Sx:=Trim(Ole1.Вид);
  Sx:=Trim(Ole1.Тип);
  Sx:=Trim(Ole1.Длина);
  Sx:=Trim(Ole1.Точность);
  Sx:=Trim(Ole1.Периодический);
  Sx:=Trim(Ole1.Сортировка);
  Sx:=Trim(Ole1.Отбор);
  Sx:=Trim(Ole1.Использование);
  Sx:=Trim(Ole1.ИзменяетсяДокументами);
  Sx:=Trim(Ole1.РучноеИзменение);
  Sx:=Trim(Ole1.Неотрицательный);
  Sx:=Trim(Ole1.РазделятьТриады);
  Sx:=Trim(Ole1.Комментарий);
  //--------------------------------
  ...
end;
```

В листинге 4.62 представлен фрагмент функции, считывающей основные свойства документов заданного вида.

**Листинг 4.62. Фрагмент функции, считывающей свойства документов заданного вида**

```
function TForm1.GetStructDocuments(Lines: TStrings): integer;
begin
  Result:=0;
  ...
  // Получаем ссылку на объект типа "Документ" метаданных
  // по его порядковому номеру
  Ole0:=Ole_1C_App.Метаданные.Документ(iDoc*1);
  sDoc:=Trim(Ole0.Идентификатор);
  sDoc:=Trim(Ole0.Синоним);
```

```
  sDoc:=Trim(Ole0.Комментарий);
  sDoc:=Trim(Ole0.ОперативныйУчет);
  sDoc:=Trim(Ole0.ПериодичностьНомера);
  sDoc:=Trim(Ole0.ДлинаНомера);
  sDoc:=Trim(Ole0.ТипНомера);
  sDoc:=Trim(Ole0.Автонумерация);
  sDoc:=Trim(Ole0.КонтрольУникальности);
  sDoc:=Trim(Ole0.БухгалтерскийУчет);
  sDoc:=Trim(Ole0.Расчет);
  sDoc:=Trim(Ole0.СоздаватьОперацию);
  sDoc:=Trim(Ole0.АвтонумерацияСтрок);
  sDoc:=Trim(Ole0.АвтоудалениеДвижений);
  sDoc:=Trim(Ole0.РедактированиеОпераций);
  sDoc:=Trim(Ole0.РазрешитьПроведение);
  sDoc:=Trim(Ole0.ОснованиеДляЛюбогоДокумента);
  ...
end;
```

В листинге 4.63 представлен фрагмент функции, считывающей список документов, вводимых на основании документов заданного вида.

**Листинг 4.63. Фрагмент функции, считывающей список документов, вводимых на основании документов заданного вида**

```
function TForm1.GetStructDocuments(Lines: TStrings): integer;
begin
  Result:=0;
  ...
  // Получаем ссылку на объект типа "Документ" метаданных по его номеру
  Ole0:=Ole_1C_App.Метаданные.Документ(iDoc*1);
  ...
  //---------------------------------------
  // Считываем документы, вводимые на основании
  CountAttr:=Ole0.ВводимыеНаОснованииДокументы.Количество;
  if CountAttr>0 then begin
     for iAttr:= 1 to CountAttr do
      begin
sDoc:=Ole0.ВводимыеНаОснованииДокументы.Получить(iAttr).Идентификатор;
```

```
Sx:=Ole0.ВводимыеНаОснованииДокументы.Получить(iAttr).Представление;
       Application.ProcessMessages;
     end;
  end;
  //----------------------------------------
  ...
end;
```

В листинге 4.64 представлен фрагмент функции, считывающей свойства реквизитов "шапки" документов заданного вида

**Листинг 4.64. Фрагмент функции, считывающей свойства реквизитов "шапки" документов заданного вида**

```
function TForm1.GetStructDocuments(Lines: TStrings): integer;
begin
  Result:=0;
  ...
  // Получаем ссылку на объект типа "Документ" метаданных по его номеру
  Ole0:=Ole_1C_App.Метаданные.Документ(iDoc*1);
  ...
  //----------------------------------------
  // Считываем свойства реквизитов "шапки" документа
  sDoc:=Trim(Ole0.РеквизитШапки(iAttr).Идентификатор);
  sDoc:=Trim(Ole0.РеквизитШапки(iAttr).Тип);
  sDoc:=Trim(Ole0.РеквизитШапки(iAttr).Вид);
  sDoc:=Trim(Ole0.РеквизитШапки(iAttr).Длина);
  sDoc:=Trim(Ole0.РеквизитШапки(iAttr).Точность);
  sDoc:=Trim(Ole0.РеквизитШапки(iAttr).НеОтрицательный);
  sDoc:=Trim(Ole0.РеквизитШапки(iAttr).РазделятьТриады);
  //----------------------------------------
  ...
end;
```

Точно так же считываются свойства реквизитов табличной части документа. Только вместо метода `РеквизитШапки` (листинг 4.64) нужно обращаться к методу `РеквизитТабличнойЧасти`.

В листинге 4.65 представлен фрагмент функции, считывающей основные свойства регистров.

**Листинг 4.65. Фрагмент функции, считывающей свойства регистра**

```
function TForm1.GetStructRegisters(Lines: TStrings): integer;
begin
 Result:=0;
 ...
 // Получаем ссылку на объект типа "Регистр" метаданных по его номеру
 Ole0:=Ole_1C_App.Метаданные.Регистр(iC*1);
 ...
 //---------------------------------------
 // Считываем основные свойства регистров
 Sx:=Trim(Ole0.Синоним);
 Sx:=Trim(Ole0.ТипРегистра);
 Sx:=Trim(Ole0.БыстраяОбработкаДвижений);
 Sx:=Trim(Ole0.МиграцияЗапрещена);
 Sx:=Trim(Ole0.Комментарий);
 //---------------------------------------
 ...
end;
```

В листинге 4.66 представлен фрагмент функции, считывающей основные свойства измерения регистра.

**Листинг 4.66. Фрагмент функции, считывающей свойства измерения регистра**

```
function TForm1.GetStructRegisters(Lines: TStrings): integer;
begin
 Result:=0;
 ...
 // Получаем ссылку на объект типа "Регистр" метаданных по его номеру
 Ole0:=Ole_1C_App.Метаданные.Регистр(iC*1);
 ...
 //---------------------------------------
 // Считываем свойства измерения регистра по его номеру
 Sx:=Trim(Ole0.Измерение(iA*1).Идентификатор);
 Sx:=Trim(Ole0.Измерение(iA*1).Синоним);
 Sx:=Trim(Ole0.Измерение(iA*1).Тип);
 Sx:=Trim(Ole0.Измерение(iA*1).Вид);
 Sx:=Trim(Ole0.Измерение(iA*1).Длина);
```

```
 Sx:=Trim(Ole0.Измерение(iA*1).Точность);
 Sx:=Trim(Ole0.Измерение(iA*1).Неотрицательный);
 Sx:=Trim(Ole0.Измерение(iA*1).РазделятьТриады);
 Sx:=Trim(Ole0.Измерение(iA*1).ОтборДвижений);
 Sx:=Trim(Ole0.Измерение(iA*1).ОтборИтогов);
 Sx:=Trim(Ole0.Измерение(iA*1).Комментарий);
 //-------------------------------------
 ...
end;
```

В листинге 4.67 представлен фрагмент функции, считывающей основные свойства ресурса регистра.

**Листинг 4.67. Фрагмент функции, считывающей свойства ресурса регистра**

```
function TForm1.GetStructRegisters(Lines: TStrings): integer;
begin
 Result:=0;
 ...
 // Получаем ссылку на объект типа "Регистр" метаданных по его номеру
 Ole0:=Ole_1C_App.Метаданные.Регистр(iC*1);
 ...
 //-------------------------------------
 // Считываем свойства ресурса регистра по его номеру
 Sx:=Trim(Ole0.Ресурс(iA*1).Идентификатор);
 Sx:=Trim(Ole0.Ресурс(iA*1).Синоним);
 Sx:=Trim(Ole0.Ресурс(iA*1).Тип);
 Sx:=Trim(Ole0.Ресурс(iA*1).Вид);
 Sx:=Trim(Ole0.Ресурс(iA*1).Длина);
 Sx:=Trim(Ole0.Ресурс(iA*1).Точность);
 Sx:=Trim(Ole0.Ресурс(iA*1).Неотрицательный);
 Sx:=Trim(Ole0.Ресурс(iA*1).РазделятьТриады);
 Sx:=Trim(Ole0.Ресурс(iA*1).Комментарий);
 //-------------------------------------
 ...
end;
```

В листинге 4.68 представлен фрагмент функции, считывающей основные свойства реквизита регистра.

**Листинг 4.68. Фрагмент функции, считывающей свойства реквизита регистра**

```
function TForm1.GetStructRegisters(Lines: TStrings): integer;
begin
 Result:=0;
 ...
 // Получаем ссылку на объект типа "Регистр" метаданных по его номеру
 Ole0:=Ole_1C_App.Метаданные.Регистр(iC*1);
 ...
 //-------------------------------------
 // Считываем свойства реквизита регистра
 Sx:=Trim(Ole0.Реквизит(iA*1).Идентификатор);
 Sx:=Trim(Ole0.Реквизит(iA*1).Синоним);
 Sx:=Trim(Ole0.Реквизит(iA*1).Тип);
 Sx:=Trim(Ole0.Реквизит(iA*1).Вид);
 Sx:=Trim(Ole0.Реквизит(iA*1).Длина);
 Sx:=Trim(Ole0.Реквизит(iA*1).Точность);
 Sx:=Trim(Ole0.Реквизит(iA*1).Неотрицательный);
 Sx:=Trim(Ole0.Реквизит(iA*1).РазделятьТриады);
 Sx:=Trim(Ole0.Реквизит(iA*1).Сортировка);
 Sx:=Trim(Ole0.Реквизит(iA*1).Комментарий);
 //-------------------------------------
 ...
end;
```

В листинге 4.69 представлен фрагмент функции, считывающей основные свойства константы.

**Листинг 4.69. Фрагмент функции, считывающей свойства константы**

```
function TForm1.GetStructConstants(Lines: TStrings): integer;
begin
 Result:=0;
 ...
 // Получаем ссылку на объект типа "Константа" метаданных по его номеру
 Ole0:=Ole_1C_App.Метаданные.Константа(iConst*1);
 ...
 //-------------------------------------
```

```
 // Считываем свойства константы
 Sx:=Trim(Ole0.Идентификатор);
 Sx:=Trim(Ole0.Синоним);
 Sx:=Trim(Ole0.Периодический);
 Sx:=Trim(Ole0.Тип);
 Sx:=Trim(Ole0.Вид);
 Sx:=Trim(Ole0.Длина);
 Sx:=Trim(Ole0.Точность);
 Sx:=Trim(Ole0.НеОтрицательный);
 Sx:=Trim(Ole0.РазделятьТриады);
 Sx:=Trim(Ole0.ОбластьРаспространения);
 Sx:=Trim(Ole0.Комментарий);
 //----------------------------------------
 ...
end;
```

В листинге 4.70 представлен фрагмент функции, считывающей информационную структуру перечислений. Этот тип агрегатных данных нужно рассмотреть более детально.

Во-первых, здесь есть некоторые нюансы, которых не было в рассмотренных ранее объектах. Во-вторых, в *части II* нам предстоит обращаться к базе данных, минуя программу 1С:Предприятие. Как отмечалось в *разд. 4.1.1*, перечисления хранятся в файле 1Cv7.MD, информацию из которого нельзя считать какими-либо стандартными способами. А в таблицах базы данных (в данном случае в DBF-файлах) значения перечислений представлены внутренними (системными) идентификаторами, алгоритм генерации которых не документирован. Эти идентификаторы являются, мягко говоря, "загадочными" (см. рис. 4.11).

В этом разделе мы рассмотрим возможность создания соответствий между "штатными" идентификаторами значений перечислений (листинг 4.70) и системными идентификаторами этих же значений.

**Листинг 4.70. Фрагмент функции, считывающей идентификаторы значений перечисления**

```
function GetStructEnum(Lines: TStrings): integer;
begin
 Result:=0;
 ...
```

```
 // Определяем количество перечислений в конфигурации
 CountEnum:=Ole_1C_App.Метаданные.Перечисление*1;
 if CountEnum>0 then begin
   // Организуем цикл с перебором всех перечислений
   iEnum:=0;
   while iEnum<CountEnum do
    begin
    iEnum:=iEnum+1;
    //-------------------------------------
    // Получаем ссылку на объект типа "Регистр" метаданных
    // по его порядковому номеру в конфигурации
    Ole0:=Ole_1C_App.Метаданные.Перечисление(iEnum*1);
    // Считываем идентифкатор перечисления
    idEnum:=Trim(Ole0.Идентификатор);
    //-------------------------------------
    //-------------------------------------
    // Получаем ссылку на заданное идентифкатором
    // перечисление как объект,
    // принадлежащий глобальному контексту конфигурации
    Ole1:=Ole_1C_App.Перечисление.ПолучитьАтрибут(''+idEnum);
    //-------------------------------------
    //-------------------------------------
    // Теперь вычисляем количество значений данного перечисления,
    //обращаясь к его соответсвующим методам
    CountVal:= Ole1.КоличествоЗначений*1;
    //-------------------------------------
    //-------------------------------------
    // Организуем цикл для перебора всех значений перечисления
    for iVal:=1 to CountVal do
     begin
     // Теперь можно использовать методы конкретного значения
     // как объекта и прочитать его "штатный" идентификатор
     idVal:=Trim(Ole1.ЗначениеПоНомеру(iVal*1).Идентификатор);
    end;
    //-------------------------------------
   end;
 end;
 ...
end;
```

Поясним текст функции `GetStructEnum` (листинг 4.70). Здесь следует отметить, что система 1С:Предприятие позволяет, с одной стороны, обратиться к перечислению как объекту метаданных, а с другой стороны — как к объекту глобального контекста задачи. То есть среди свойств объекта `Метаданные` системы 1С:Предприятия есть объект типа `Перечисление`. Поэтому мы можем получить как количество всех перечислений, так и полный список их идентификаторов (как мы делали это со справочниками, документами, регистрами).

Но объект `Перечисление` является также частью глобального контекста программы 1С:Предприятие. И в этом качестве этот объект имеет метод `ПолучитьАтрибут`, позволяющий получить доступ к объекту `Перечисление конкретного вида` по его идентификатору. Объект `Перечисление конкретного вида` обладает такими свойствами, как перечень значений (каждое из которых также является объектом) и методами `КоличествоЗначений`, `ЗначениеПоНомеру` и `ЗначениеПоИдентификатору`.

У каждого значения перечисления (как объекта) есть собственные методы `Вид`, `ПредставлениеВида`, `Выбран`, `ПорядковыйНомер`, `Идентификатор`.

В строке

```
Ole0:=Ole_1C_App.Метаданные.Перечисление(iEnum*1);
```

листинга 4.70 мы получаем ссылку на *объект метаданных* типа `Перечисление` по его порядковому номеру в конфигурации.

Затем, используя соответствующий метод объекта, получаем его идентификатор:

```
idEnum:= Trim(Ole0.Идентификатор);
```

Далее обращаемся к объекту `Перечисление` *глобального контекста* конфигурации по идентификатору и получаем ссылку на перечисление конкретного вида:

```
Ole1:=Ole_1C_App.Перечисление.ПолучитьАтрибут(''+idEnum);
```

Теперь мы можем использовать методы перечисления (как объекта) для того, чтобы получить список идентификаторов его значений. Сначала получим количество значений перечисления:

```
CountVal:= Ole1.КоличествоЗначений*1;
```

А затем (в цикле) получаем ссылку на конкретное значение (как объект) перечисления и (обратившись к его соответствующему методу) его идентификатор:

```
idVal:=Trim(Ole1.ЗначениеПоНомеру(iVal*1).Идентификатор);
```

Эту строку можно для ясности разбить на две строки:

```
// Получаем ссылку на объект "Значение перечисления"
Ole2:= Ole1.ЗначениеПоНомеру(iVal*1);
// Обращаемся к методу "идентификатор" этого объекта
idVal:=Trim(Ole2.Идентификатор);
```

Скомпилируем наш проект, запустим его на выполнение и нажмем кнопку **Структура**. На рис. 4.176 представлен результат считывания информационной структуры "нашей" конфигурации.

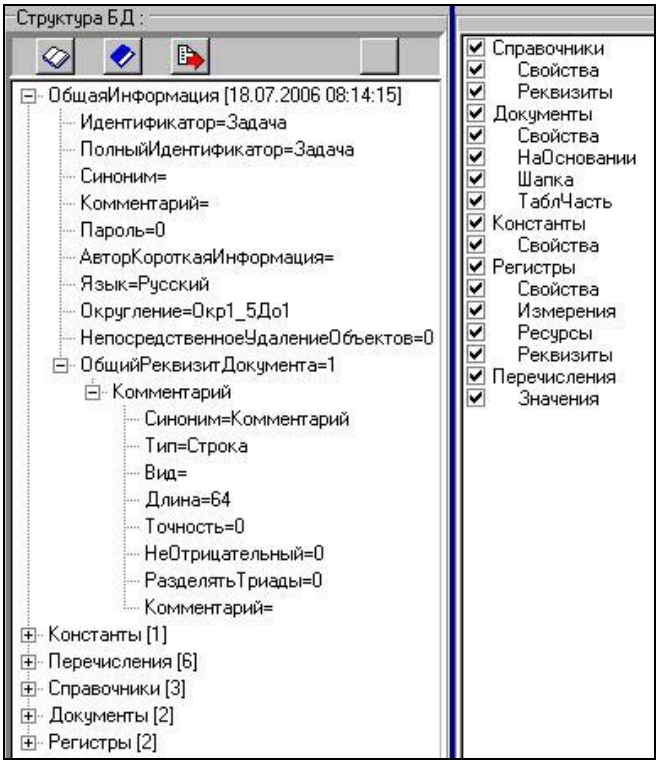



**Рис. 4.176.** Результат считывания информационной структуры "нашей" конфигурации

Общие свойства конфигурации не исчерпываются набором свойств, значения которых здесь представлены. Но в рамках данного раздела перед нами не стоит задача разработать полномасштабное программное средство для работы с информационной структурой системы 1С:Предприятие.

На рис. 4.177 представлена часть информационной структуры справочника `ФизЛица` (свойства справочника).

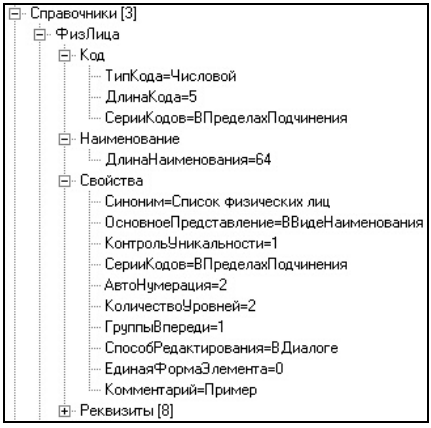



**Рис. 4.177.** Часть информационной структуры справочника `ФизЛица`. Свойства справочника

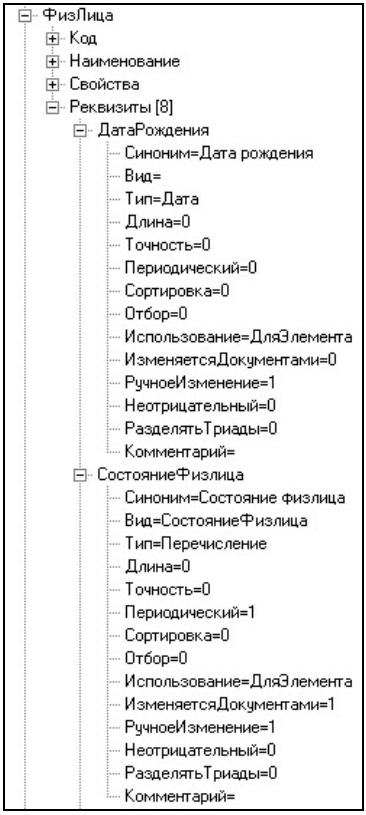



**Рис. 4.178.** Часть информационной структуры справочника `ФизЛица`. Свойства реквизитов справочника

Как видите, здесь отражены все те настройки справочника `ФизЛица`, что мы сделали в предыдущих разделах.

На рис. 4.178 представлена часть информационной структуры справочника `ФизЛица` (свойства реквизитов справочника).
Аналогично представлена информационная структура по документам и регистрам.

На рис. 4.179 представлена информационная структура перечислений "нашей" конфигурации.

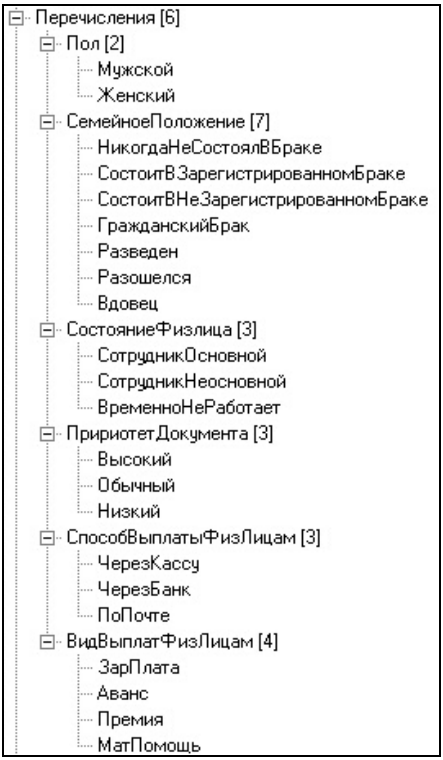



**Рис. 4.179.** Информационная структура перечислений "нашей" конфигурации

В папке \Part_1\Chapter_04\Topic_4_4 на сопроводительном компакт-диске книги есть два текстовых файла:

- 1C_example_DB_structure_00.TXT — информационная структура, выгруженная "штатными" средствами системы 1С:Предприятие;
- 1C_example_DB_structure_01.TXT — информационная структура, полученная в результате наших экспериментов.

Здесь следует отметить, что мы (в рамках этого раздела) не ставили перед собой цель получить исчерпывающую информацию о структуре конфигурации, поэтому сравнивать оба эти файла следует с известной долей понимания.

Вернемся к вопросу идентификации значений перечислений вне контекста системы 1С:Предприятие.

Как отмечалось ранее, при использовании механизма OLE Automation считываются те же ("штатные") идентификаторы значений перечислений, что были определены при конфигурировании системы, тогда как в таблицах базы данных хранятся системные (внутренние) идентификаторы значений перечислений. Это налагает определенные трудности при считывании информации из таблиц базы данных вне контекста системы 1С:Предприятие.

Попробуем решить эту проблему комбинированным способом. То есть при считывании информационной структуры с использованием механизма OLE Automation установим соответствие между "штатным" идентификатором значения перечисления и его системным идентификатором, а затем будем использовать это соответствие при доступе к базе данных вне контекста системы 1С:Предприятие (в *части II*).

Среди средств встроенного языка программирования системы 1С:Предприятие есть функция `ЗначениеВСтрокуВнутр` (или `ValueToStringInternal`). Ее назначение — преобразование значения объекта агрегатного типа из внутреннего представления в строковое. Как следует из описания функции, строковое представление агрегатного типа данных имеет специальный системный формат. Данный системный формат использует *внутренний системный идентификатор* данных. Функция принимает один параметр, который может принимать значения любого типа данных (как базовых, так и агрегатных), кроме тех типов данных, которые не имеют собственно значения (например, `Периодический`, `Таблица` и т. д.).

Попробуем применить эту функцию в рамках нашей задачи, т. е. к агрегатному типу данных `Перечисление`. Вспомним, что в явном виде (используя механизм OLE Automation) мы не можем обмениваться объектами агрегатного типа с системой 1С:Предприятие. Следовательно (как в случае с регистрами, см. ранее), нам нужна функция (в глобальном модуле конфигурации), на вход которой можно было бы передать идентификатор перечисления и порядковый номер значения этого перечисления, а она должна обеспечить соответствующие преобразования. Эта функция должна располагаться в глобальном модуле конфигурации и быть доступной для внешних приложений (т. е. должна быть с ключевым словом `Экспорт`).

Откроем Конфигуратор системы 1С:Предприятия и создадим в глобальном модуле функцию (листинг 4.71).

**Листинг 4.71. Функция в глобальном модуле, обеспечивающая вызов функции ЗначениеВСтрокуВнутр для объектов типа Значение_перечисления**

```
// Функция, которая находит у заданного перечисления значение по номеру
// и вызывает для найденного значения функцию "ЗначениеВСтрокуВнутр".
// Здесь:
//  идПеречисления — идентификатор перечисления (вид),
//  НомерЗначения  — номер значения перечисления.
Функция гл9_ЗначениеВСтрокуВнутр_для_Перечисления(идПеречисления,
НомерЗначения) Экспорт
 id = "";
 НомЗнач = ЧИСЛО(НомерЗначения);
 Если НомЗнач>0 Тогда
   G = Перечисление.ПолучитьАтрибут(""+СокрЛП(идПеречисления));
   Колво = G.КоличествоЗначений();
   Если НомЗнач<=Колво Тогда
     Объект = G.ЗначениеПоНомеру(НомЗнач*1);
     id = ЗначениеВСтрокуВнутр(Объект);
   КонецЕсли;
 КонецЕсли;
 Возврат id;
КонецФункции
```

Текст этой функции достаточно прост и не нуждается в дополнительных комментариях. Все методы (кроме ЗначениеВСтрокуВнутр) мы уже использовали в этом разделе.

Попробуем обратиться к функции гл9_ЗначениеВСтрокуВнутр_для_Перечисления из нашего проекта. Для этого разместим в форме проекта какую-нибудь кнопку (SpeedButton7) и напишем обработчик события "нажатие этой кнопки" (листинг 4.72).

**Листинг 4.72. Обработчик события "нажатие кнопки вызова функции гл9_ЗначениеВСтрокуВнутр_для_Перечисления"**

```
procedure TForm1.SpeedButton7Click(Sender: TObject);
Var
 Sx: string;
 Ole1: OleVariant;
 Num: integer; // Номер тестируемого значения перечисления
 idEnum, // Идентификатор тестируемого перечисления
```

```
 idVal, // "Штатный" идентификатор тестируемого значения перечисления
 //----------------------
 // Результат, который возвращает функция
 // "гл9_ЗначениеВСтрокуВнутр_для_Перечисления"
 idValSys: string;
 //----------------------
begin
  Ole_1C_App:=ConnectTo1C(Edit_1Cinit.Text);
  if not VarIsEmpty(Ole_1C_App) then begin
     // Идентификатор тестируемого перечисления
     idEnum:='СостояниеФизлица';
     // Номер тестируемого значения
     Num:=1;
     // Получаем ссылку на перечисление как объект
     // глобального контекста программы 1С:Предприятие
     Ole1:=Ole_1C_App.Перечисление.ПолучитьАтрибут(''+idEnum);
     // Получаем "штатный" идентификатор значения перечисления
     idVal:=Trim(Ole1.ЗначениеПоНомеру(Num*1).Идентификатор);
     // Формируем строку для вызова функции
     // "гл9_ЗначениеВСтрокуВнутр_для_Перечисления"
     Sx:= 'гл9_ЗначениеВСтрокуВнутр_для_Перечисления("'
        + idEnum
        + '",'
        + IntToStr(Num)
        + ')';
     // Вызываем функцию
     //"гл9_ЗначениеВСтрокуВнутр_для_Перечисления"
     idValSys:=Trim(Ole_1C_App.EvalExpr(Sx);
     // Результат выводим на экран в виде сообщения
     ShowMessage('Перечисление='
               + idEnum
               + '; НомерЗначения='
               + IntToStr(Num)
               + '; id='
               + idVal
               + ';  ЗначениеВСтрокуВнутр='
               + idValSys);
  end;
end;
```

Результат, возвращаемый функцией `гл9_ЗначениеВСтрокуВнутр_для_Перечисления` для значения номер `1` (`СотрудникОсновной`) перечисления `СостояниеФизлица`, приведен на рис. 4.180.

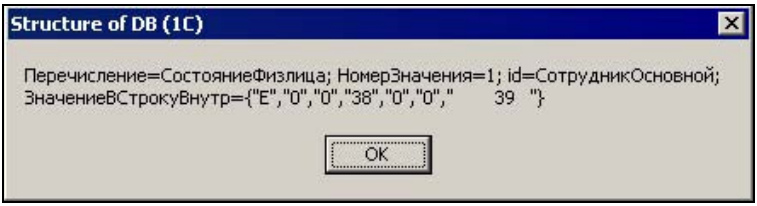



**Рис. 4.180.** Результат, возвращаемый функцией `гл9_ЗначениеВСтрокуВнутр_для_Перечисления`

Если сделать соответствующие изменения в тексте функции `GetStructEnum` (см. листинг 4.70) и вновь прочитать информационную структуру перечислений, то получим результат, приведенный на рис. 4.181.

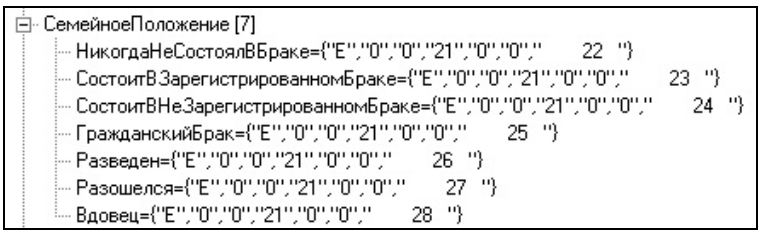



**Рис. 4.181.** Результат , возвращаемый функцией `гл9_ЗначениеВСтрокуВнутр_для_Перечисления` для всех значений перечислений

Если внимательно посмотреть на рис. 4.181, то видно, что системный идентификатор располагается в самом конце строки (последний "член" в строке), обрамленной фигурными скобками. Наша задача — выделить его и конвертировать соответствующим образом.

Вновь запустим конфигуратор системы 1С:Предприятие и в глобальном модуле "нашей" конфигурации создадим еще три функции (листинги 4.73—4.75).

**Листинг 4.73. Функция глобального модуля, инвертирующая строку символов**

```
// Инвертирование строки символов
Функция гл9_ИнвертироватьСтроку(ИсхСтрока)
  Перем Res; // Строка
  Перем i;
```

```
  Res="";
  Если СтрДлина(ИсхСтрока)>0 Тогда
    i=0;
    Пока i<СтрДлина(ИсхСтрока) Цикл
      i=i+1;
      // Считываем очередной символ из строки
      S1=Сред(ИсхСтрока,i,1);
      Res=S1+Res; // Инвертируем
    КонецЦикла;
  КонецЕсли;
  Возврат Res;
КонецФункции
```

Функция `гл9_ИнвертироватьСтроку` (листинг 4.73) просто "переворачивает" строку символов (первый символ строки становится последним, а последний — первым). Ее текст достаточно прост и не требует пояснений.

**Листинг 4.74. Функция глобального модуля, возвращающая системный идентификатор объекта**

```
// Получение внутреннего (системного) идентификатора объекта
Функция гл9_Объект_ПолучитьSysID(ТестируемыйОбъект)  Экспорт
  id = ""; // Здесь будет системный идентификатор
  // Преобразуем значение объекта агрегатного типа
  // из внутреннего представления в строковое
  S=ЗначениеВСтрокуВнутр(ТестируемыйОбъект);
  // Заменяем двойные кавычки на пробелы
  S = СтрЗаменить(S, Симв(34), " ");
  // Заменяем правые фигурные скобки на пробелы
  // и удаляем конечные пробелы из строки
  S = СокрЛП(СтрЗаменить(S, "}", " "));
  // Инвертируем строку (последний "член" становится первым)
  S = гл9_ИнвертироватьСтроку(S);
  // Получаем номер позиции первой запятой в инвертированной строке
  НомерПозицииЗапятой = Найти(S,",");
  Если НомерПозицииЗапятой>1 Тогда
    // Если есть запятая, выделяем все, что расположено слева до нее,
    // удаляем конечные пробелы и опять инвертируем строку.
```

```
    // Это и будет самый последний "член"
    S = гл9_ИнвертироватьСтроку(СокрЛП(Лев(S, НомерПозицииЗапятой-1)));
    // Конвертируем его в системный идентификатор объекта
    id = _IdToStr(S);
  КонецЕсли;
  Возврат id;
КонецФункции
```

Функция гл9_Объект_ПолучитьSysID (листинг 4.74) получает на входе объект, преобразует его значение из внутреннего представления в строковое, затем выделяет последний "член" из строки и конвертирует его в системный (внутренний) идентификатор.

На основе текста функции гл9_ЗначениеВСтрокуВнутр_для_Перечисления (листинг 4.71) создадим функцию, которая будет получать в качестве входных параметров идентификатор перечисления и номер значения перечисления, а возвращать — системный (внутренний) идентификатор. Текст этой функции приведен в листинге 4.75.

**Листинг.4.75. Функция глобального модуля, возвращающая системный идентификатор значения перечисления**

```
// Получение системного идентификатора значения перечисления
Функция гл9_Перечисление_Получить_SysID_
ЗначенияПоНомеру(идПеречисления,НомерЗначения) Экспорт
  // идПеречисления — идентификатор тестируемого перечисления
  // НомерЗначения  — номер тестируемого значения перечисления
  id = "";
  НомЗнач = ЧИСЛО(НомерЗначения);
  Если НомЗнач>0 Тогда
    G = Перечисление.ПолучитьАтрибут(""+СокрЛП(идПеречисления));
    Колво = G.КоличествоЗначений();
    Если НомЗнач<=Колво Тогда
      Объект = G.ЗначениеПоНомеру(НомЗнач*1);
      id = гл9_Объект_ПолучитьSysID(Объект);
    КонецЕсли;
  КонецЕсли;
  Возврат id;
КонецФункции
```

Сохраним изменения, сделанные в конфигурации, вернемся к нашему Delphi-проекту и внесем небольшое изменение в текст обработчика нажатия кнопки `SpeedButton7` (см. листинг 4.72).

Для этого две строки:

```
//"гл9_ЗначениеВСтрокуВнутр_для_Перечисления"
Sx:= 'гл9_ЗначениеВСтрокуВнутр_для_Перечисления("'
```

заменим двумя другими:

```
//"гл9_Перечисление_Получить_SysID_ЗначенияПоНомеру"
Sx:= 'гл9_Перечисление_Получить_SysID_ЗначенияПоНомеру ("'
```

Скомпилируем проект и запустим на выполнение. По нажатию кнопки `SpeedButton7` на экране появится сообщение, показанное на рис. 4.182.

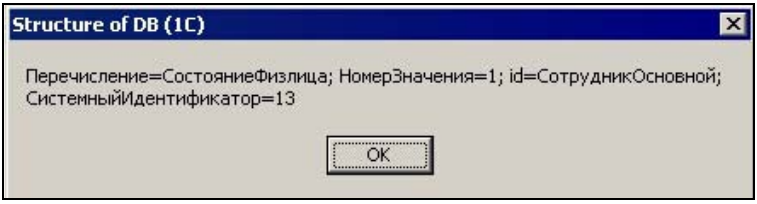



**Рис. 4.182.** Результат , возвращаемый функцией `гл9_Перечисление_Получить_SysID_ЗначенияПоНомеру`

Теперь внесем изменения в функцию `GetStructEnum` (см. листинг 4.70), чтобы дополнительно прочитать значения системных (внутренних) идентификаторов значений перечислений (листинг 4.76).

**Листинг 4.76. Фрагмент текста функции `GetStructEnum`**

```
function TForm1.GetStructEnum(Lines: TStrings): integer;
begin
  Result:=0;
  ...
  for iVal:=1 to CountVal do
   begin
    idVal:=Trim(Ole1.ЗначениеПоНомеру(iVal*1).Идентификатор);
    Sx:= 'гл9_Перечисление_Получить_SysID_ЗначенияПоНомеру ("'
        + idEnum
        + '",'
        + IntToStr(iVal)
```

```
        + ')';
    idValSys:=Trim(Ole_1C_App.EvalExpr(Sx);
    Lines.Add(#9+#9+idVal+'='+idValSys);
  end;
  ...
end;
```

На рис. 4.183 представлен результат считывания информационной структуры с учетом сделанных изменений.

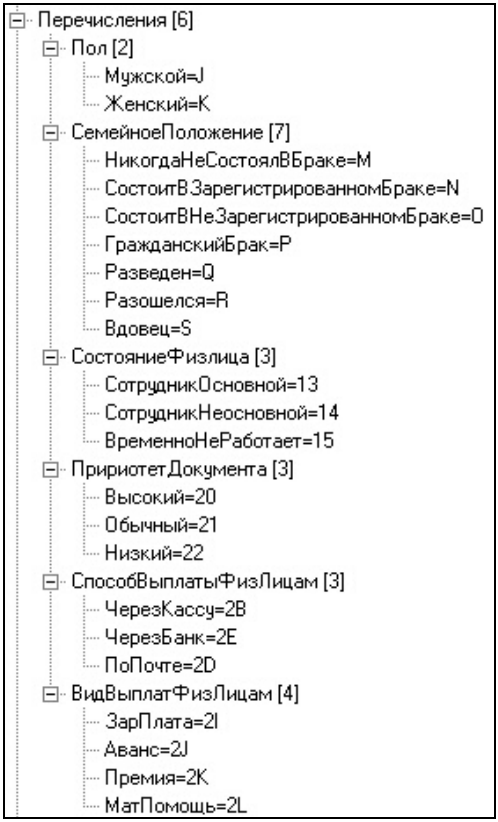



**Рис. 4.183.** Результат считывания информационной структуры перечислений с учетом системных идентификаторов значений перечислений

Сопоставив информацию, представленную на рис. 4.183, с рис. 4.4, 4.10 и 4.11, можно сделать уже вполне определенные, полезные выводы. Предоставлю это читателю (в *части II* мы вернемся к этому вопросу).

***Примечание***

В папке \Part_1\Chapter_04\Topic_4_4 сопроводительного компакт-диска книги есть текстовый файл 1C_example_DB_structure_02.TXT, содержащий информационную структуру, фрагмент которой представлен на рис. 1.183.

Исходные тексты этого примера (проект sp_LoadStructure_1C.DPR) содержатся на сопроводительном компакт-диске книги в папке: \Part_1\Chapter_04\Topic_4_4\sp_LoadStructure_1C.PAS.

***Примечание***

Напомню, что представленные выше примеры исходных текстов — это упрощенная иллюстрация к рассматриваемым вопросам. Назначение этих примеров — дать представление о возможном подходе при решении задач информационного обмена с системой 1С:Предприятие. При работе с реальными базами 1С может потребоваться их дальнейшая доработка.

## 4.4.2. Doc-файлы (хранилища структурированной информации). Файл конфигурации системы 1С:Предприятие — 1Cv7.MD

Doc-файл (DocFile, compound file, составной файл, хранилище структурированной информации, structured storage) — это средство для хранения структурированной информации в виде единого файла. Структура файла организована в виде дерева по аналогии с файловой системой. При этом эквивалентом каталога (папки) является хранилище, а эквивалентом файла — поток. Doc-файлы ориентированы на хранение разнородных данных (текст, видео, графика, звук и т. д.). Doc-файлы широко используются в среде Windows для хранения наборов свойств OLE, составных документов и контейнеров OLE. В качестве примера можно привести такие приложения, как MS Word, MS Excel, MS PowerPoint и др.

Физически Doc-файл организован по кластерному типу (наподобие файловой системы FAT). Первые 4 байта Doc-файла: `D0h`, `CFh`, `11h`, `E0h` (по этому признаку можно визуально оценить, является ли данный файл Doc-файлом).

Более подробно организация Doc-файлов описана, например, в статьях:

- Климентьев К. Е. "Внутренний формат документов MS Word" (**http://www.uinc.ru/articles/39/**);
- Новодворский С. "Доступ к MD-файлам при помощи VBA" (**http://www.klerk.ru/soft/1c/?2429**, **http://www.hare.ru/kb/know-how/article.html?id=59**).

Логически Doc-файл состоит из следующих элементов.

- *Хранилище* (storage) — аналог каталогов файловой системы. Здесь хранятся ссылки на информационные "потоки". Как и любая древообразная структура, Doc-файл имеет корневой каталог (root storage).
- *Поток* (stream) — структурированная информация (аналог файлов).
- *Блокированные данные* (lock-bytes) — неструктурированная информация.

> **Примечание**
>
> Создание Doc-файлов сложной структуры и считывание информации из них подробно описаны в специальной литературе.

В системе 1С:Предприятие эта технология используется для хранения информационной структуры (метаданные) каждой конфигурации (базы данных) в файле 1Cv7.MD, который представляет собой Doc-файл (хранилище структурированной информации).

Рассмотрим процесс считывания информационной структуры базы данных 1С:Предприятие из файла 1Cv7.MD на простом примере.

Запустим Delphi и создадим новый проект с именем Read_from_1Cv7_MD.DPR.

> **Примечание**
>
> Исходные тексты этого примера есть на сопроводительном компакт-диске книги в папке \Part_1\Chapter_04\Topic_4_4\Read_from_1Cv7_MD_00.PAS.

На рис. 4.184 представлена форма проекта.

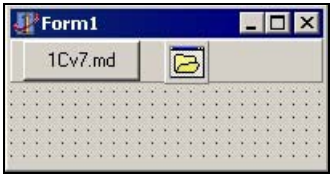



**Рис. 4.184.** Форма Delphi-проекта Read_from_1Cv7_MD.DPR

В листинге 4.77 представлен текст обработчика события "нажатие кнопки **1Cv7.md**".

**Листинг 4.77. Контроль файла с помощью функции `StgIsStorageFile`**

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  OD1.FileName:=fn_1Cv7_md;
```

```
  OD1.InitialDir:=ExtractFilePath(fn_1Cv7_md);
  if OD1.Execute then begin
     fn_1Cv7_md:=OD1.FileName;
     Caption:=fn_1Cv7_md;
     if StgIsStorageFile(PWideChar(fn_1Cv7_md)) = S_OK then begin
        ShowMessage(ExtractFileName(fn_1Cv7_md)+' — это Doc-файл');
     end
     else begin
        ShowMessage(ExtractFileName(fn_1Cv7_md)+' — это не Doc-файл');
     end;
  end;
end;
```

Здесь после того, как файл был выбран, с помощью функции `StgIsStorageFile` проверяется — является ли выбранный файл Doc-файлом. Результаты тестирования этой функции представлены на рис. 4.185 и 4.186.

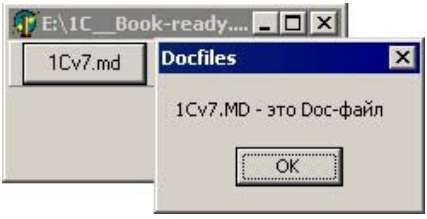



**Рис. 4.185.** Тестирование функции `StgIsStorageFile` (положительный результат проверки)

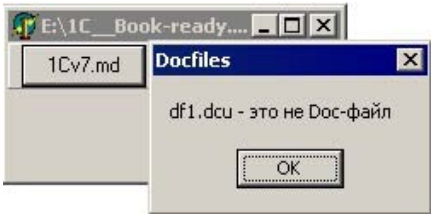



**Рис. 4.186.** Тестирование функции `StgIsStorageFile` (отрицательный результат проверки)

Добавим в форму проекта Delphi-компонент `TTreeView` (рис. 4.187).

Изменим текст обработчика события "нажатие кнопки **1Cv7.md**" следующим образом (листинг 4.78).

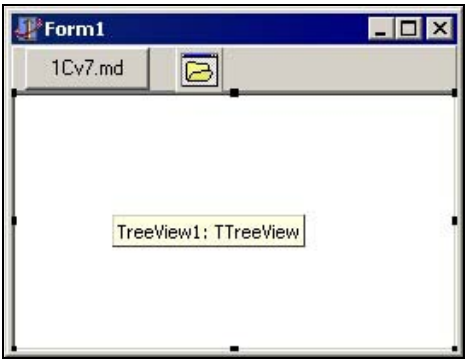



**Рис. 4.187.** Форма Delphi-проекта Read_from_1Cv7_MD.DPR

**Листинг 4.78. Обработчик события "нажатие кнопки *1Cv7.md*"**

```
procedure TForm1.Button1Click(Sender: TObject);
Var
  Node: TTreeNode;
  stgRoot: IStorage;
begin
  OD1.FileName:=fn_1Cv7_md;
  OD1.InitialDir:=ExtractFilePath(fn_1Cv7_md);
  if OD1.Execute then begin
     fn_1Cv7_md:=OD1.FileName;
     Caption:=fn_1Cv7_md;
     if StgIsStorageFile(PWideChar(fn_1Cv7_md))=S_OK then begin
      // ShowMessage(ExtractFileName(fn_1Cv7_md)+' — это Doc-файл');
        OleCheck(StgOpenStorage(PWideChar(fn_1Cv7_md),
                                nil,
                                stgm_read or stgm_share_exclusive,
                                nil,
                                0,
                                stgRoot));
        TreeView1.Items.BeginUpdate;
        try
          TreeView1.Items.Clear;
          Node:=TreeView1.Items.Add(nil,'<Root>');
          GetNextStorage(Node,stgRoot);
          Node.Expand(true);
```

```
        finally
          TreeView1.Items.EndUpdate;
          stgRoot:=nil;
        end;
      end
      else begin
        ShowMessage(ExtractFileName(fn_1Cv7_md)+' — это не Doc-файл');
      end;
  end;
end;
```

Здесь Doc-файл открывается для чтения с помощью функции `StgOpenStorage`. В случае успеха в переменной `stgRoot: IStorage;` возвращается указатель на интерфейс корневого каталога (root storage) открытого хранилища информации.

С помощью методов интерфейса `IStorage` (каталога, по аналогии с файловой системой) можно прочитать список его элементов, которые представляют собой "подкаталоги" и "файлы" (по аналогии с файловой системой).

Таким образом, комбинируя методы `OpenStorage` и `EnumElements` интерфейса `IStorage`, можно "пролистать" всю структуру Doc-файла. В листинге 4.79 приведен текст рекурсивной функции `GetNextStorage`, которая выполняет эти действия.

**Листинг 4.79. Функция `GetNextStorage` проекта**

```
procedure TForm1.GetNextStorage(pn: TTreeNode; stg: IStorage);
Var
  ESS: IEnumStatStg;
  StatStg: TStatStg;
  NodeName: string;
  cn: TTreeNode;
  ctgChild: IStorage;
  Sx: string;
begin
  OleCheck(stg.EnumElements(0,nil,0,ESS));
  while ESS.Next(1,StatStg,nil) = S_OK do
   begin
     NodeName:=StatStg.pwcsName;
     Sx:='???';
     if StatStg.dwType = STGTY_STORAGE then Sx:='STORAGE';
```

```
      if StatStg.dwType = STGTY_STREAM then Sx:='STREAM';
      if StatStg.dwType = STGTY_PROPERTY then Sx:='PROPERTY';
      if StatStg.dwType = STGTY_LOCKBYTES then Sx:='LOCKBYTES';
      cn:=TreeView1.Items.AddChild(pn,NodeName+'='+Sx+'');
      if (StatStg.dwType = STGTY_STORAGE) then begin
         OleCheck(Stg.OpenStorage(StatStg.pwcsName,
                  nil,
                  stgm_read or stgm_share_exclusive,
                  nil,
                  0,
                  ctgChild));
         GetNextStorage(cn,ctgChild);
      end;
   end;
end;
```

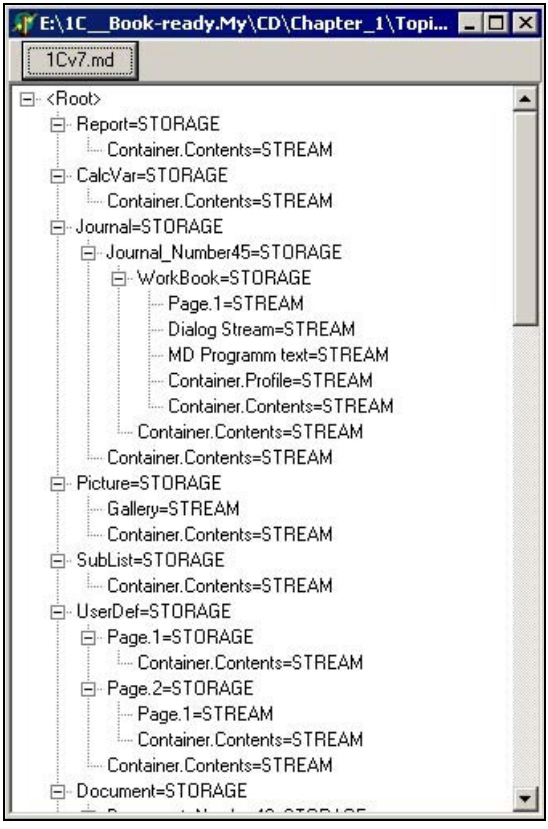



**Рис. 4.188.** Результат считывания структуры файла 1Cv7.MD "нашей" конфигурации

***Примечание***

Подробное описание назначения соответствующих методов интерфейсов `IStorage` и `IEnumStatStg` можно найти в специальной литературе.

На рис. 4.188 представлен результат считывания информационной структуры Doc-файла (1Cv7.MD) "нашей" конфигурации.

***Примечание***

Исходные тексты этого примера есть на сопроводительном компакт-диске книги в папке \Part_1\Chapter_04\Topic_4_4\Read_from_1Cv7_MD_01.PAS.

Модифицируем наш проект, добавив на форму Delphi-компонент типа `TMemo` (рис. 4.189).

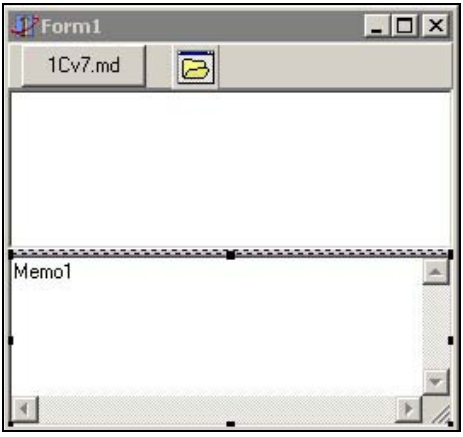



**Рис. 4.189.** Форма Delphi-проекта Read_from_1Cv7_MD.DPR

Добавим обработчик события "нажатие левой кнопки мыши на компоненте `TTreeView`" (листинг 4.80).

**Листинг 4.80. Обработчик события "нажатие левой кнопки мыши на компоненте `TTreeView`"**

```
procedure TForm1.TreeView1Click(Sender: TObject);
Var
  ParentNodeName,
  NodeName: WideString;
  Sx: string;
  DocStorage,
```

```
  Stg:IStorage;
  Strm: IStream;
  DataStream: TOleStream;
begin
  Memo1.Lines.Clear;
  if TreeView1.Selected<>NIL then begin
     NodeName:=TreeView1.Selected.Text;
     Sx:=Trim(sp_GetWordFromString('=', NodeName,2));
     if Sx= 'STREAM' then begin // Выбранный узел должен быть
                                // потоком("файлом")
        ParentNodeName:=TreeView1.Selected.Parent.Text;
        Sx:=Trim(sp_GetWordFromString('=', ParentNodeName,2));
        // Родитель выбранного узла должен быть хранилищем ("каталогом")
        if (Sx= 'STORAGE') or (Sx= '') then begin
        // Открываем Doc-файл (1Cv7.MD)
           OleCheck(StgOpenStorage(PWideChar(fn_1Cv7_md),
                                   nil,
                                   stgm_read or STGM_DIRECT
                                     or stgm_share_exclusive,
                                   nil,
                                   0,
                                   Stg));
           try
          // Определяем имя родителя выбранного узла TTreeView
          // (первое слово в имени узла). Это должен быть "каталог"
       ParentNodeName:=Trim(sp_GetWordFromString('=', ParentNodeName,1));
         // Открываем "каталог" в хранилище с именем ParentNodeName
             if Stg.OpenStorage(POleStr(ParentNodeName),
                     nil,
                     stgm_read or STGM_DIRECT or STGM_SHARE_EXCLUSIVE,
                     nil,
                     0,
                     DocStorage) <> S_OK then Exit;
          // Определяем имя выбранного узла TTreeView
          // (первое слово в имени узла). Это должен быть поток("файл")
             NodeName:=Trim(sp_GetWordFromString('=', NodeName,1));
             if DocStorage.OpenStream(POleStr(NodeName),
                 nil,
                 stgm_read or STGM_DIRECT or STGM_SHARE_EXCLUSIVE,
```

```
                0,
                Strm) <> S_OK then Exit;
          //----------------------------
          // Считываем данные из потока ("файла")
             DataStream:=TOleStream.Create(Strm);
             try
               SetLength(Sx, DataStream.Size);
               DataStream.ReadBuffer(Pointer(Sx)^, DataStream.Size);
               Memo1.Lines.Add(Sx);
             finally
               DataStream.free;
               DataStream:=NIL;
               Strm:=nil;
             end;
         //----------------------------
           finally
             stg:=nil;
           end;
        end
        else begin
           Memo1.Lines.Add(NodeName+': ожидалось, что родитель STORAGE');
        end;
     end
     else begin
        Memo1.Lines.Add(NodeName+': нужно выбрать STREAM');
     end;
  end;
end;
```

Здесь определяется и запоминается имя выбранного узла (`NodeName`) компонента `TTreeView` и его родителя (`ParentNodeName`). Затем открываются Doc-файл (функция `StgOpenStorage`), "каталог" с именем `ParentNodeName` (метод `Stg.OpenStorage`) и поток (`NodeName`) "каталога" `ParentNodeName` (метод `DocStorage.OpenStream`). Из потока `NodeName` считываются данные (с помощью методов объекта `DataStream`). На рис. 4.190 представлен результат считывания данных из потока `Main MetaData Stream` файла 1Cv7.MD. Такое построение текста обработчика связано с тем, что Doc-файл открывается только на период считывания структуры файла и только при считывании данных из выбранного потока.

**Рис. 4.190.** Результат считывания информации из потока `Main MetaData Stream` файла 1Cv7.MD

Здесь следует отметить, что такой способ считывания информации (листинг 4.80):

```
SetLength(Sx, DataStream.Size);
DataStream.ReadBuffer(Pointer(Sx)^, DataStream.Size);
Memo1.Lines.Add(Sx);
```

может не сработать в том случае, если в потоке есть символы, "усекающие" строку.

Более надежный способ (хотя и более медленный):

```
Var
  ...
  Yes: byte;
```

```
  i: integer;
  Buffer: PChar;
begin
  ...
    GetMem(Buffer, DataStream.Size);
    try
      DataStream.ReadBuffer(Buffer^,DataStream.Size);
      Sx:='';
      for i:=0 to (DataStream.Size-1) do
       begin
         Yes:=0;
         // Пропускаем только "читабельные" символы
         if (ORD(CHAR(Buffer[i])) in [32..255]) then Yes:=1;
         if (ORD(CHAR(Buffer[i])) in [9,10,13]) then Yes:=1;
         if Yes>0 then begin
            Sx:=Sx+CHAR(Buffer[i]);
         end;
      end;
      Memo1.Lines.TEXT:=Sx;
    finally
       FreeMem(Buffer);
    end;
  ...
end;
```

**Примечание**

Исходные тексты этого примера приведены на сопроводительном компакт-диске книги в папке \Part_1\Chapter_04\Topic_4_4\Read_from_1Cv7_MD_02.PAS.

Как видите, в потоке `Main MetaData Stream` файла 1Cv7.MD хранятся данные об информационной структуре конфигурации базы данных 1С:Предприятие. В данном случае — информационная структура "нашей" конфигурации.

**Примечание**

Полное содержание потока приведено на сопроводительном компакт-диске книги в текстовом файле \Part_1\Chapter_04\Topic_4_4\1Cv7_md_Main_MetaData_Stream.TXT.

Даже простейший визуальный анализ данных из потока `Main MetaData Stream` (файл 1Cv7_md_Main_MetaData_Stream.TXT) и данных из файла 1Cv7.DD по-

казывает (например, для справочника ФизЛица) следующие закономерности (табл. 4.8).

***Таблица 4.8.*** *Результаты сравнения словаря данных(1Cv7.DD) и потока данных* `Main MetaData Stream` *для справочника* ФизЛица

| Категория | Поток **Main MetaData Stream** | Файл 1Cv7.DD |
|---|---|---|
| Идентификатор справочника | **12**<br>Первый член в строке:<br>{**"12"**,"ФизЛица","Пример", "Список физических лиц"...} | **SC12** (имя таблицы)<br>Строка:<br>T=**SC12** \| Справочник ФизЛица |
| Идентификатор реквизита:<br>ДатаРождения | **15**<br>Первый член в строке:<br>{**"15"**,"ДатаРождения","", "Дата рождения","D",...} | **SP15** (имя поля)<br>Строка:<br>F=**SP15**\| (P)ДатаРождения\|D... |
| Идентификатор реквизита:<br>Пол | **29**<br>Первый член в строке:<br>{"29","Пол","","Пол","E","0" ,"0","17"...} | **SP29** (имя поля)<br>Строка:<br>F=**SP29**\|(P)Пол\|C |
| Идентификатор реквизита:<br>ДомашнийАдрес | **16**<br>Первый член в строке:<br>{**"16"**,"ДомашнийАдрес","", "Домашний адрес","S","64"...} | **SP16** (имя поля)<br>Строка:<br>F=**SP16**\| (P)ДомашнийАдрес\|C\|64 |

Если проанализировать восьмой член в строках потока `Main MetaData Stream`, то видно, что его значение "чудесным образом" "коррелируется" со значением первого члена строк потока. Например, для регистра УчетИзменений СостоянияФизЛиц:

```
{"52","УчетИзмененийСостоянияФизЛиц", ...
```

в строке:

```
{"53","ФизЛицо","","","B","0","0","12","0","0","0","0"},
```

значение восьмого члена равно **12**, а это соответствует первому члену строки:

```
{"12","ФизЛица","Пример",...
```

Эта информация будет крайне полезной в том случае, если доступ к базе данных 1С:Предприятие осуществляется вне контекста программы 1С, т. е. при прямых SQL-запросах к таблицам базы данных. Нам эта информация потребуется в *части II* данной книги.

**Примечание**

Файл 1Cv7.MD содержит значительно больше полезной информации, чем мы здесь охватили. Более подробную информацию о структуре файла 1Cv7.MD можно получить, например, на сайтах Интернета (табл 4.10 — далеко не полный список).

***Таблица 4.9**. Интернет-ресурсы, где можно получить информацию о файле 1Cv7.MD*

| Сайт | Адрес |
|---|---|
| Гендин.ru "1cv7.MD. Knowledge Base::База Знаний" | **http://gendin.ru/md/index.html** |
| Персональная страница Федора Езеева | **http://1c.alterplast.ru/1cv7md/** |
| 1L:Project. Open Source | **http://1l.w4b.ru/** |
| Доступ к данным V7 из других систем. Исследование MD | **http://1c.proclub.ru/modules/kb/print.php?storyid=18** |
| "Structured Storage". The binary structure of Ole/Compound Documents (структура Doc-файлов на низком уровне) | **http://user.cs.tu-erlin.de/%7Eschwartz/pmh/guide.html** |
| "Внутренний формат документов MS WORD", статья | **http://www.uinc.ru/articles/39/** |
| "Доступ к MD-файлам при помощи VBA", статья | **http://www.klerk.ru/soft/1c/?2429, http://www.hare.ru/kb/know-how/article.html?id=59** |
| Доступ к данным V7 из других систем. Исследование MD | **http://www.erp-volga.com/hare/kb/article.8.html** |

**Примечание**

Знания о структуре файла 1Cv7.MD, доступные сейчас, — результат работы многих людей, сделавших это на общественных началах и безвозмездно. Поэтому список сайтов по данной тематике может быть существенно расширен. Приношу глубочайшие извинения тем авторам сайтов (по данной тематике), которые внесли свой вклад в данный процесс, но не попали в эту таблицу.

Существуют программные модули, ориентированные на работу с Doc-файлами и, в частности, с файлом 1Cv7.MD. Например (неполный список):

- DocFile Browser is plugin for FAR Manager. DocFile Browser allows browsing, viewing and changing contents of compound files;
- compound.dll — внешний компонент для 1С:Предприятие 7.7 (предназначен для работы с compound файлами *.ERT, *.MD);

- gcomp.EXE — инструмент для преобразования файла конфигурации 1С (1Cv7.MD) в набор текстовых файлов и обратно;
- MDtoText.EXE — программа "Конвертор модулей метаданных", предназначена для автоматического извлечения текстов модулей из файла метаданных 1С:Предприятия во внешние файлы и обратного преобразования информации из внешних файлов в файл метаданных (1Cv7.MD);
- ExtractMd.EXE — распаковщик MD-файла (создает в текущем каталоге набор файлов/каталогов — точно воспроизводящий все содержимое MD);
- mdxtractor.EXE — распаковщик MD-файла (разворачивает дерево метаданных).

Все эти программы имеют статус freeware (бесплатные) и доступны в Интернете.

***Примечание***

Приношу глубочайшие извинения другим разработчикам ПО (freeware), которые внесли свой вклад в этот процесс, но их программы не попали в этот список.

***Примечание***

Напомню, что приведенные в книге примеры исходных текстов — это упрощенная иллюстрация к рассматриваемым вопросам. Назначение этих примеров — дать представление о возможном подходе при решении задач информационного обмена с системой 1С:Предприятие. При работе с реальными базами 1С может потребоваться их дальнейшая доработка.

## 4.5. Метод *EvalExpr*

Система 1С:Предприятие (как сервер OLE Automation) предоставляет полный доступ к своему глобальному контексту. Поэтому объект OLE-сервер 1С:Предприятие в качестве своих методов может иметь системные процедуры и функции, а также процедуры и функции глобального программного модуля, объявленные с ключевым словом `Экспорт`. Кроме того, у OLE-сервера 1С:Предприятие есть три дополнительных метода: `Initialize`, `EvalExpr`, `ExecuteBatch`.

Ранее мы уже обращались к методу `EvalExpr` для решения ряда задач. В этом разделе метод `EvalExpr` будет рассмотрен более детально.

Метод `EvalExpr` позволяет вычислить переданное в параметре выражение на встроенном языке системы 1С:Предприятие. Синтаксис:

`EvalExpr(`*<Параметр>*`)`

Здесь *<Параметр>* — выражение, записанное на встроенном языке 1С:Предприятие (тип — `Строка`). Возвращаемое значение — результат вычисленного выражения (число, строка, дата или значение любого агрегатного типа данных). Результат с неопределенным типом данных преобразуется к строковому типу.

Создадим новый Delphi-проект с именем sp_EvalExpr_1C.DPR.

Функционально программа sp_EvalExpr_1C.EXE должна обеспечивать соединение с системой 1С:Предприятие и вызов метода `EvalExpr` с заданным параметром. Результат, возвращаемый методом `EvalExpr`, должен конвертироваться в тип `String` и сохраняться в буфере.

Форма проекта представлена на рис. 4.191.

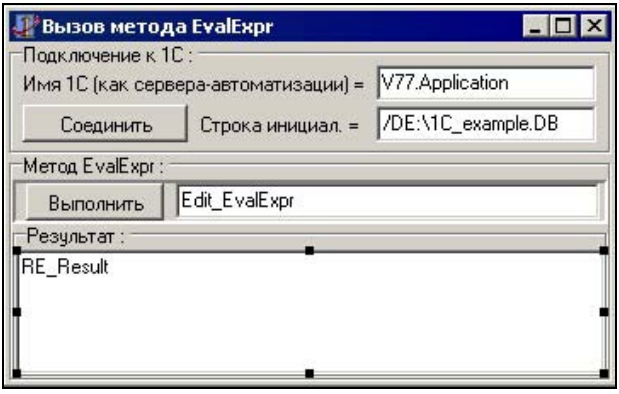



**Рис. 4.191.** Форма проекта sp_EvalExpr_1C.DPR

Как видите, здесь нет ничего нового по сравнению с тем, что мы делали ранее. По нажатию кнопки **Соединить** производятся действия, подробно рассмотренные в *главе 3*. По нажатию кнопки **Выполнить** вызывается метод `EvalExpr` с параметром, значение которого хранится в `Edit_EvalExpr.Text`. Результат сохраняется в `RE_Result.Lines`.

Текст обработчика события "нажатие кнопки **Выполнить**" приведен в листинге 4.81.

**Листинг 4.81. Обработчик события "нажатие кнопки *Выполнить*"**

```
procedure TForm1.SpeedButton2Click(Sender: TObject);
Var
  oleRes: OleVariant;// Результат, возвращаемый методом EvalExpr
```

```
  sRes: string;
  Sx: string;
begin
  if not VarIsEmpty(Ole_1C_App) then begin
     // Строка-параметр для метода EvalExpr
     Sx:=Trim(Edit_EvalExpr.Text);
     if length(Sx)>0 then begin
        // Вызов метода EvalExpr
        oleRes:=Ole_1C_App.EvalExpr(Sx);
        // Конвертируем результат в встроку
        sRes:=Trim(oleRes);
        RE_Result.Lines.ADD('----------------------------');
        RE_Result.Lines.ADD('EvalExpr = '+Sx);
        RE_Result.Lines.ADD('Result = '+sRes);
     end;
     Edit_EvalExpr.SetFocus;
  end
  else begin
     ShowMessage('Нет соединения с 1С');
  end;
end;
```

Текст обработчика достаточно прост и пояснений не требует.

> ***Примечание***
>
> Исходный текст проекта sp_EvalExpr_1C.DPR приведен на сопроводительном компакт-диске книги в папке \Part_1\Chapter_04\Topic_4_5\sp_EvalExpr_1C.PAS.

Прежде чем запустить проект sp_EvalExpr_1C.DPR на выполнение, откроем Конфигуратор системы 1С:Предприятие и в глобальном модуле создадим две функции, текст которых приведен в листингах 4.82 и 4.83.

**Листинг 4.82. Первая простая функция в глобальном модуле для тестирования метода EValExpr**

```
Функция гл9_А_плюс_Б(А,Б)  Экспорт
  Возврат А+Б;
КонецФункции
```

**Листинг 4.83. Вторая простая функция в глобальном модуле для тестирования метода `EValExpr`**

```
Функция гл9_ах_плюс_b(a,x,b)  Экспорт
  Возврат a*x+b;
КонецФункции
```

Как видите, текст обеих функций прост. Каждая из них выполняет простейшую математическую операцию.

Сохраним конфигурацию (см. рис. 2.17).

Теперь запустим наш проект на выполнение, введем в `Edit_EvalExpr.Text` значение параметра для `EvalExpr`:

```
гл9_А_плюс_Б(5, 7)
```

и нажмем кнопку **Выполнить**.

Повторим то же самое для другого значения параметра:

```
гл9_ах_плюс_b(2, 0.5, 2)
```

Результат представлен на рис. 4.192 и комментариев не требует.

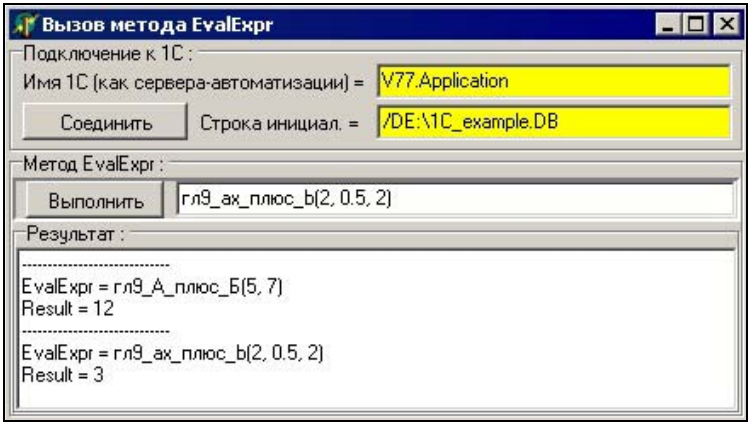



**Рис. 4.192.** Результат выполнения функций глобального модуля, вызов которых производился с использованием метода `EvalExpr`

В табл. 4.10 приведены другие примеры вызова некоторых функций системы 1С:Предприятие (с использованием метода `EvalExpr`) и результаты их выполнения.

***Таблица 4.10.*** *Примеры использования метода* `EvalExpr`

| Значение параметра | Результат выполнения |
|---|---|
| СтрЗаменить("В этой строке заменить В на в", "В", "в") | в этой строке заменить в на в |
| Сред("Привет, как дела ?????", 8,11) | как дела ? |
| КодСимв("*") | 42 |
| Симв(42) | * |
| Макс(5, 1, 7, 3, 2) | 7 |
| Мин(5, 1, 7, 3, 2) | 1 |
| Шаблон("Сост.физ.лица: [Перечисление. СостояниеФизлица. СотрудникНеосновной]") | Сост.физ.лица: Сотрудник по неосновному месту работы |
| РабочаяДата() | 22.07.2006 |
| ДобавитьМесяц("22.07.2006",6) | 22.01.2007 |
| КонМесяца("22.07.2006") | 31.07.2006 |
| НачНедели("22.07.2006") | 17.07.2006 |
| КонНедели("22.07.2006") | 23.07.2006 |
| НомерНеделиГода("22.07.2006") | 30 |
| ПериодСтр("01.01.2006", "31.03.2006") | 1 Квартал 2006 г. |
| КаталогИБ() | E:\1C_example.DB\ |
| КаталогПрограммы() | F:\1Cv77\BIN\ |
| КаталогВременныхФайлов() | C:\DOCUME~1\Sergey\LOCALS~1\Temp\ |
| МонопольныйРежим() | 0 |
| ОсновнойЯзык() | 1<br>Число 1 — если основной язык — русский;<br>Число 0 — если основной язык — английский. |
| ЗапуститьПриложение("Notepad.exe ИмяФайла.txt") | Запускается внешняя программа (в данном случае Notepad.exe) |
| ПолучитьДатуТА() | 15.07.2006 |
| ПолучитьВремяТА() | 12:26:28 |
| ПолучитьТА() | 15.07.06 12:26:28 |

В предыдущих разделах мы столкнулись с тем, что при считывании информации из базы данных системы 1С:Предприятие нужно предпринимать дополнительные меры, чтобы конвертировать значения агрегатного типа в строку (например, в *разд. 4.1.4*, листинг 4.12). Эту проблему можно решить проще, если создать в глобальном модуле соответствующую функцию, которая будет преобразовывать значение переданного ей параметра (любого типа) в строку. Текст функции приведен в листинге 4.84.

**Листинг 4.84. Преобразование переданного значения в тип `Строка`**

```
// Получить строковое представление объекта
// для вызова через EvalExpr
// с использованием механизма OLE Automation
Функция гл9_ОбъектКонвертироватьВСтроку(Объект)  Экспорт
  Возврат СокрЛП(Объект);
КонецФункции
```

Текст функции `гл9_ОбъектКонвертироватьВСтроку` достаточно прост, пояснения не требуются. В листинге 4.85 приведены примеры вызова этой функции, а на рис. 4.193 — результат выполнения.

**Листинг 4.85. Тестирование функции `гл9_ОбъектКонвертироватьВСтроку`**

```
гл9_ОбъектКонвертироватьВСтроку("325.75")
гл9_ОбъектКонвертироватьВСтроку("12.01.2006")
гл9_ОбъектКонвертироватьВСтроку(Перечисление.СемейноеПоложение.
   НикогдаНеСостоялВБраке)
```

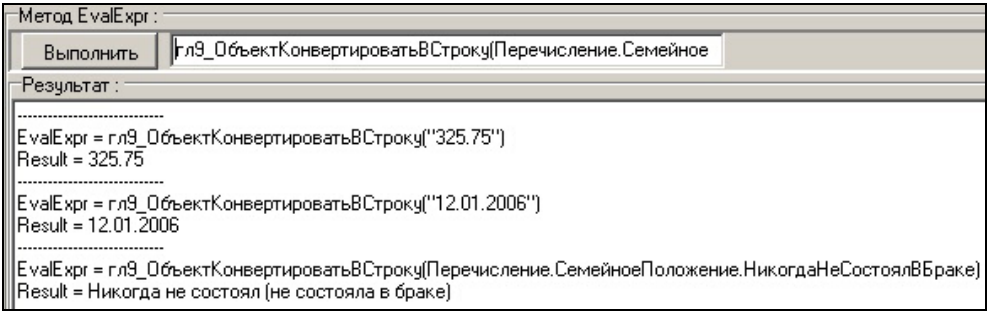



**Рис. 4.193.** Результат выполнения функций глобального модуля `гл9_ОбъектКонвертироватьВСтроку`

Во всех приведенных примерах использования метода `EvalExpr` в качестве параметра передается значение (но не переменная). Кроме этого, каждый вызов метода `EvalExpr` производится вне зависимости от предыдущих вызовов. Это существенное ограничение при его практическом использовании. Чтобы решить эту проблему, объявим в глобальном модуле "нашей" конфигурации переменную (массив длиной 100) с ключевым словом `Экспорт` (листинг 4.86).

**Листинг 4.86. Переменная (массив) в глобальном модуле для использования в методах `EvalExpr` и `ExecuteBatch`**

```
// Для внешних обращений через OLE (методы EvalExpr и ExecuteBatch)
Перем гл9БуферДляOLE[100] Экспорт;
```

Теперь мы можем вызывать метод `EvalExpr` с сохранением результата в элементе массива `гл9БуферДляOLE`, а во всех последующих вызовах метода `EvalExpr` использовать значение этого элемента массива в виде входного параметра. В листинге 4.87 приведен простой пример, иллюстрирующий этот способ.

**Листинг 4.87. Пример использования переменной `гл9БуферДляOLE` (тип `Число`)**

```
ВвестиЧисло(гл9БуферДляOLE[1], "Введите какое-нибудь число", 12, 2, 60)
гл9_ОбъектКонвертироватьВСтроку(гл9БуферДляOLE[1])
```

Результат вызова метода `ВвестиЧисло` приведен на рис. 4.194.

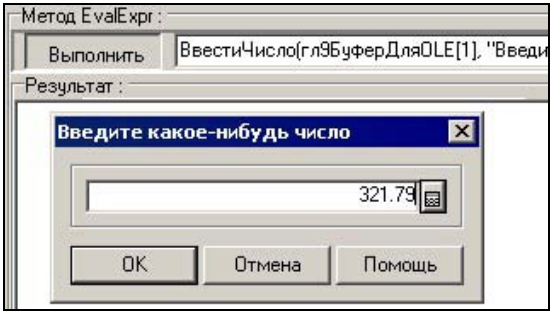



**Рис. 4.194.** Вызов метода `ВвестиЧисло` с использованием метода `EvalExpr` и переменной `гл9БуферДляOLE`

***Примечание***

Напомню, что диалоговые окна, которые вызываются при обращении к программе 1С:Предприятие (в качестве сервера OLE Automation), могут быть скрыты окнами других приложений. Для их "поиска" можно использовать сочетание клавиш <Alt>+<Tab>.

Результат выполнения команд (листинг 4.87) приведен на рис. 4.195.

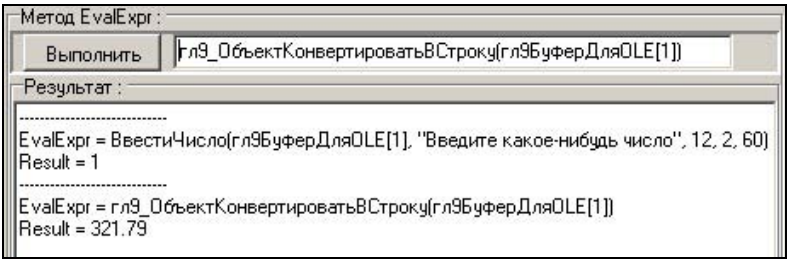



**Рис. 4.195.** Результат выполнения команд, приведенных в листинге 4.87

Точно так же можно использовать переменную `гл9БуферДляOLE` для хранения значений агрегатного типа. В листинге 4.88 приведены команды, иллюстрирующие выбор значения типа `Справочник` и сохранение этого значения в элементе массива `гл9БуферДляOLE`.

**Листинг 4.88. Пример использования переменной `гл9БуферДляOLE` (агрегатный тип `Справочник`)**

```
ВвестиЗначение(гл9БуферДляOLE[2],"Выберите Физ.Лицо из списка",
    "Справочник.ФизЛица")
гл9_ОбъектКонвертироватьВСтроку(гл9БуферДляOLE[2])
```

Результат выполнения команды `ВвестиЗначение` представлен на рис. 4.196.

На экране появилось диалоговое окно для выбора значения агрегатного типа `Справочник` (вид `ФизЛица`).

Результат выполнения команд (листинг 4.88) приведен на рис. 4.197.

Особенностью метода `EvalExpr` является отсутствие возможности выполнения команд вроде следующей:

```
EvalExpr("A = 5 + 7")
```

Это может быть существенным ограничением при выполнении "цепочки" последовательных вызовов метода `EvalExpr`. Например, нет возможности выполнить метод `СоздатьОбъект`. Эту проблему можно легко решить, создав в глобальном модуле конфигурации соответствующую функцию (листинг 4.89).

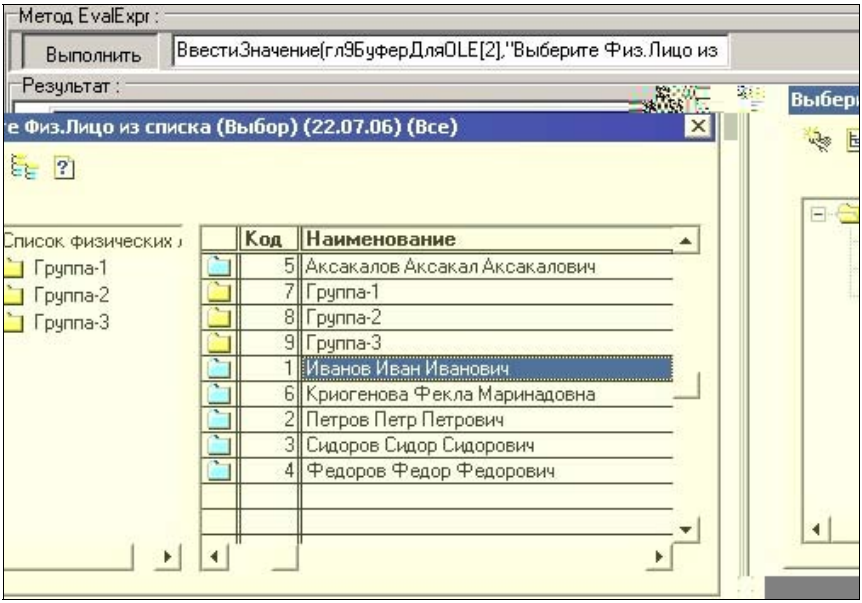



**Рис. 4.196.** Вызов метода `ВвестиЗначение` (для агрегатного типа `Справочник`) с использованием `EvalExpr` и переменной `гл9БуферДляOLE`

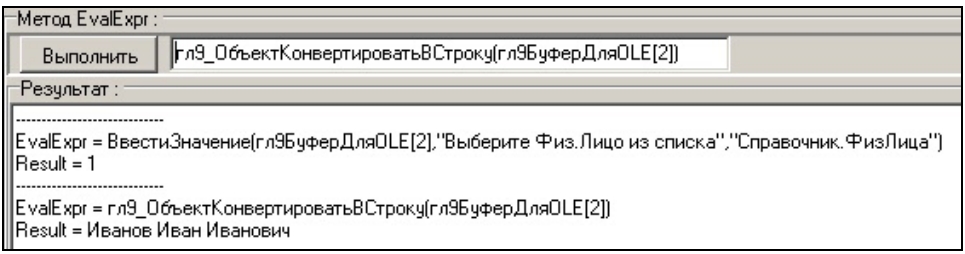



**Рис. 4.197.** Результат выполнения команд, приведенных в листинге 4.88

**Листинг 4.89. Функция в глобальном модуле, реализующая вызов метода `СоздатьОбъект`**

```
// Функция "СоздатьОбъект" для вызова через EvalExpr
// с использованием механизма OLE Automation
Функция гл9_СоздатьОбъект(ИмяОбъекта,НомерЭлементаМассива) Экспорт
  Res = 0;
  Если СтрДлина(СокрЛП(ИмяОбъекта))>0 Тогда
    КолвоЭлем=Разм(гл9БуферДляOLE);
```

```
    Если (НомерЭлементаМассива>0) И (НомерЭлементаМассива<=КолвоЭлем)
Тогда
гл9БуферДляOLE[НомерЭлементаМассива]=СоздатьОбъект(""+СокрЛП(ИмяОбъекта))
;
      Res = 1;
    КонецЕсли;
  КонецЕсли;
  Возврат Res;
КонецФункции
```

Поясним текст функции `гл9_СоздатьОбъект`.

Входные параметры функции:

- *ИмяОбъекта* — полное имя объекта (как оно объявлено в конфигурации), включая имя типа данных и идентификатор объекта (например, `Справочник.ФизЛица`);
- *НомерЭлементаМассива* — номер элемента массива `гл9БуферДляOLE`, которому будет присвоен результат функции `СоздатьОбъект`.

В строке

```
КолвоЭлем=Разм(гл9БуферДляOLE);
```

получаем размер массива (переменная `гл9БуферДляOLE`).

Далее проверяем корректность значения входной переменной `НомерЭлементаМассива`:

```
Если (НомерЭлементаМассива>0) И (НомерЭлементаМассива<=КолвоЭлем) Тогда
```

И, наконец, вызываем метод `СоздатьОбъект`. Результат сохраняем в заданном элементе массива `гл9БуферДляOLE`:

```
гл9БуферДляOLE[НомерЭлементаМассива]=СоздатьОбъект(""+СокрЛП(ИмяОбъекта));
```

В листинге 4.90 приведен пример вызова функции `гл9_СоздатьОбъект`.

**Листинг 4.90. Пример использования функции `гл9_СоздатьОбъект`**

```
EvalExpr = гл9_СоздатьОбъект("Справочник.ФизЛица", 1)
Result = 1
----------------------------
EvalExpr = ТипЗначения(гл9БуферДляOLE[1])
Result = 11
----------------------------
EvalExpr = ТипЗначенияСтр(гл9БуферДляOLE[1])
Result = Справочник
```

Здесь сначала создается объект `Справочник.ФизЛица`. Ссылка на объект сохраняется в элементе с номером **1** массива `гл9БуферДляOLE`. Затем определяется тип значения элемента с номером **1** массива `гл9БуферДляOLE`.

Попробуем решить с помощью разработанных функций глобального модуля более прикладную задачу, например, попытаемся прочитать ФИО "физического лица" с заданным кодом (аналог того, как мы это делали ранее, — листинг 4.5). В листинге 4.91 представлен перечень команд, решающих эту задачу.

**Листинг 4.91. Пример использования функции `гл9_СоздатьОбъект`: чтение из справочника**

```
гл9_СоздатьОбъект("Справочник.ФизЛица", 1)
ВвестиЧисло(гл9БуферДляOLE[2], "Введите код физ. лица", 10, 0, 60)
гл9БуферДляOLE[1].НайтиПоКоду(ЧИСЛО(гл9БуферДляOLE[2]),0)
гл9_ОбъектКонвертироватьВСтроку(гл9БуферДляOLE[1])
```

Здесь (как и в предыдущем примере) создается объект `Справочник.ФизЛица`, ссылка на который сохраняется в элементе с номером **1** массива `гл9БуферДляOLE`. Затем с помощью метода `ВвестиЧисло` вводится код "физического лица" (например 5, рис. 4.198).

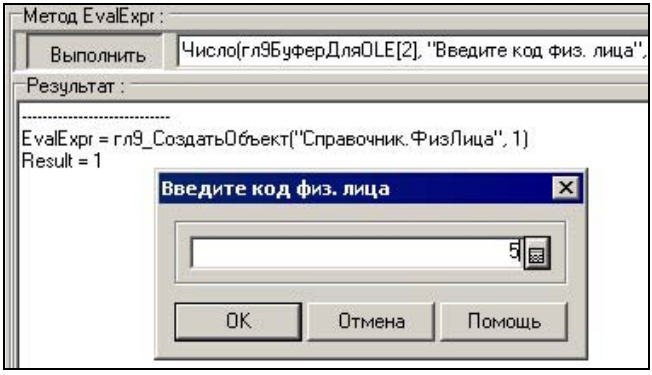



**Рис. 4.198.** Ввод кода "физического лица" (листинг 4.91)

Далее с помощью метода `НайтиПоКоду` справочника позиционируем справочник на элементе справочника с кодом 5. Затем вызываем функцию `гл9_ОбъектКонвертироватьВСтроку`. В качестве параметра передаем ей значение элемента массива `гл9БуферДляOLE[1]` (это справочник, который позиционирован на элементе с кодом 5). Результат выполнения команд (листинг 4.91) приведен на рис. 4.199.

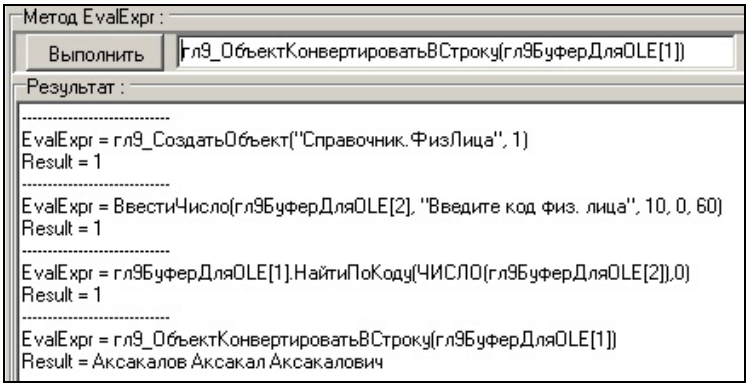



**Рис. 4.199.** Результат выполнения команд, приведенных в листинге 4.91

Посмотрим, каким образом можно использовать объект `СписокЗначений` (при обращении через метод `EvalExpr`). Добавим в глобальный модуль конфигурации еще одну функцию `гл9_СписокЗначений_ДобавитьЗначение`. Ее назначение — вызов "штатного" метода `ДобавитьЗначение` объекта `СписокЗначений`. Текст функции приведен в листинге 4.92.

**Листинг 4.92. Функция `гл9_СписокЗначений_ДобавитьЗначение`**

```
// Функция вызывает метод "ДобавитьЗначение" для объекта "СписокЗначений"
Функция гл9_СписокЗначений_ДобавитьЗначение(СписЗнач,Знач,ЗначСтр="")
    Экспорт
  res = 0;
  СписЗнач.ДобавитьЗначение(Знач,ЗначСтр);
  res = 1;
  Возврат res;
КонецФункции
```

Текст функции достаточно прост и пояснений не требует.

Теперь (используя метод `EvalExpr`) последовательно выполним следующие команды (листинг 4.93).

**Листинг 4.93. Пример использования функции `гл9_СоздатьОбъект`: `СписокЗначений`**

```
гл9_СоздатьОбъект("Справочник.ФизЛица", 1)
гл9_СоздатьОбъект("СписокЗначений", 2)
гл9БуферДляOLE[1].НайтиПоКоду(1,0)
```

```
гл9_СписокЗначений_ДобавитьЗначение(гл9БуферДляOLE[2],
гл9БуферДляOLE[1].ТекущийЭлемент())
гл9БуферДляOLE[1].НайтиПоКоду(2,0)
гл9_СписокЗначений_ДобавитьЗначение(гл9БуферДляOLE[2],
гл9БуферДляOLE[1].ТекущийЭлемент())
гл9БуферДляOLE[1].НайтиПоКоду(3,0)
гл9_СписокЗначений_ДобавитьЗначение(гл9БуферДляOLE[2],
гл9БуферДляOLE[1].ТекущийЭлемент())
гл9БуферДляOLE[2].ВыбратьЗначение(гл9БуферДляOLE[3])
гл9_ОбъектКонвертироватьВСтроку(гл9БуферДляOLE[3])
```

Здесь создаются объекты `Справочник.ФизЛица` и `СписокЗначений`. Затем справочник `ФизЛица` поочередно позициионируется на элементах справочника с кодами 1, 2 и 3. Каждый раз свойство `ТекущийЭлемент` добавляется в список значений:

```
гл9_СписокЗначений_ДобавитьЗначение(гл9БуферДляOLE[2],
    гл9БуферДляOLE[1].ТекущийЭлемент())
```

Далее вызывается метод `ВыбратьЗначение` объекта `СписокЗначений`.

В результате на экране появится диалоговое окно (рис. 4.200).

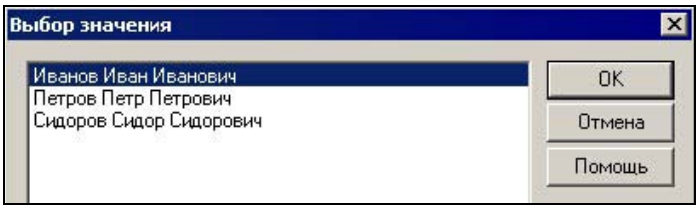



**Рис. 4.200.** Результат вызова метода `ВыбратьЗначение` объекта `СписокЗначений`

После выбора какой-либо строки из списка значений в элементе массива `гл9БуферДляOLE[3]` будет храниться ссылка на соответствующий "текущий элемент" справочника `ФизЛица`. Здесь следует отметить, что в `гл9БуферДляOLE[3]` хранится именно ссылка на объект агрегатного типа `ТекущийЭлемент` справочника. То есть в дальнейшем мы можем обращаться ко всем свойствам и методам этого объекта.

Далее вызывается функция `гл9_ОбъектКонвертироватьВСтроку`, которой в качестве входного параметра передается ссылка на выбранный `Текущий Элемент` справочника. В результате получим ФИО соответствующего "физического лица":

```
Result = Иванов Иван Иванович
```

При использовании метода `EvalExpr` и массива `гл9БуферДляOLE` может потребоваться функция, которая позволит присвоить элементу массива `гл9БуферДляOLE` заданное значение. В листинге 4.94 представлен текст функции `гл9_ПрисвоитьЗначениеЭлементуМассива`.

**Листинг 4.94. Функция, реализующая присвоение какого-то значения элементу массива `гл9БуферДляOLE`**

```
// Присвоение значения элементу массива для вызова через EvalExpr
// с использованием механизма OLE Automation
Функция гл9_ПрисвоитьЗначениеЭлементуМассива(Объект,НомерЭлементаМассива)
    Экспорт
  Res = 0;
  КолвоЭлем=Разм(гл9БуферДляOLE);
  Если (НомерЭлементаМассива>0) И (НомерЭлементаМассива<=КолвоЭлем) Тогда
    гл9БуферДляOLE[НомерЭлементаМассива] = Объект;
    Res = 1;
  КонецЕсли;
  Возврат Res;
КонецФункции
```

Посмотрим, каким образом можно использовать функцию `гл9_ПрисвоитьЗначениеЭлементуМассива` на практике. В листинге 4.95 представлен пример выбора значения перечисления.

**Листинг 4.95. Пример выбора значения перечисления с использованием функции `ВвестиПеречисление`**

```
гл9_ПрисвоитьЗначениеЭлементуМассива("СостояниеФизлица",1)
ВвестиПеречисление(гл9БуферДляOLE[1],"Выберите перечисление")
гл9_ОбъектКонвертироватьВСтроку(гл9БуферДляOLE[1])
```

По определению, в "штатную" функцию `ВвестиПеречисление` передается переменная, которая должна содержать значение типа `Перечисление` или идентификатор соответствующего перечисления. Поэтому сначала вызывается функция `гл9_ПрисвоитьЗначениеЭлементуМассива`, где элементу массива присваивается идентификатор нужного перечисления (в данном случае `СостояниеФизлица`). В результате элементу с номером **1** массива `гл9БуферДляOLE` будет присвоено значение `СостояниеФизлица` типа `Строка`.

Затем вызывается функция `ВвестиПеречисление`, куда передается идентификатор перечисления (переменная `гл9БуферДляOLE[1]`). В результате на экране появится диалоговое окно выбора значения перечисления (рис. 4.201).

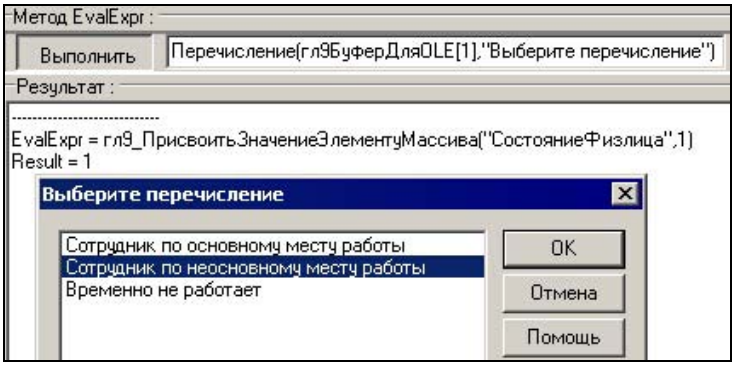



**Рис. 4.201.** Выбор значения перечисления `СостояниеФизлица`

Идентификатор перечисления можно вводить интерактивно, в диалоговом окне. Для этого вместо функции:

```
гл9_ПрисвоитьЗначениеЭлементуМассива("СостояниеФизлица",1)
```

в листинге 4.95 нужно использовать функцию:

```
ВвестиСтроку(гл9БуферДляOLE[1],"Введите идентификатор перечисления",64)
```

На рис. 4.202 представлено диалоговое окно для ввода значений типа `Строка`.

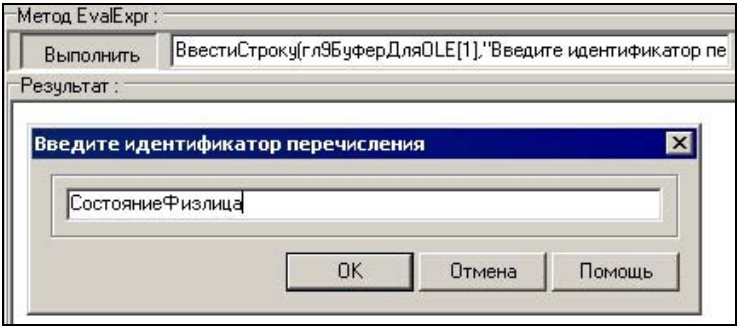



**Рис. 4.202.** Интерактивный ввод идентификатора перечисления

Значение любого типа данных, доступного в конфигурации, можно сохранить в файле (функция `ЗначениеВФайл`), а затем прочитать из файла (функция `ЗначениеИзФайла`). Так можно сохранять значения элементов массива

гл9БуферДляOLE (если в этом будет необходимость) между сеансами работы с системой 1С:Предприятие. Пример приведен в листинге 4.96.

**Листинг 4.96. Пример сохранения значений элементов массива гл9БуферДляOLE в файле**

```
...
гл9БуферДляOLE[2].ВыбратьЗначение(гл9БуферДляOLE[3])
ЗначениеВФайл("C:\1C_ЗначениеВФайл.dat",гл9БуферДляOLE[3])
...
гл9_ОбъектКонвертироватьВСтроку(ЗначениеИзФайла("C:\1C_ЗначениеВФайл.dat"))
...
```

Еще одно полезное применение метода EvalExpr может быть в том, чтобы открывать отчеты и другие внешние формы (обработки). Для этого используется функция ОткрытьФорму системы 1С:Предприятие. Но чтобы открытая форма была доступна пользователю, нужно открыть главное окно программы 1С:Предприятие. Этот вопрос мы рассмотрим в следующем разделе.

# 4.6. Метод *ExecuteBatch*

В предыдущем разделе мы подробно рассмотрели метод EvalExpr. Все бы хорошо, но каждую строку команды приходится выполнять отдельно кроме того, невозможно использовать управляющие конструкции и операторы. В этом разделе мы рассмотрим метод ExecuteBatch, свободный от этих недостатков.

Метод ExecuteBatch позволяет выполнить заданную входным параметром последовательность операторов на встроенном языке системы 1С:Предприятие. Синтаксис:

ExecuteBatch(*<СтрокаОператоров>*)

Параметр: *<СтрокаОператоров>* — строковое выражение (текст программы на встроенном языке 1С:Предприятие). Метод возвращает значение логического типа: TRUE — если последовательность операторов выполнена успешно; FALSE — если нет.

***Примечание***

В OLE Automation TRUE и FALSE имеют соответственно значения –1 (минус единица) и 0.

На основе предыдущего проекта (sp_EvalExpr_1C.DPR) создадим новый Delphi-проект с именем sp_ExecuteBatch_1C.dpr. Форма проекта представлена на рис. 4.203.

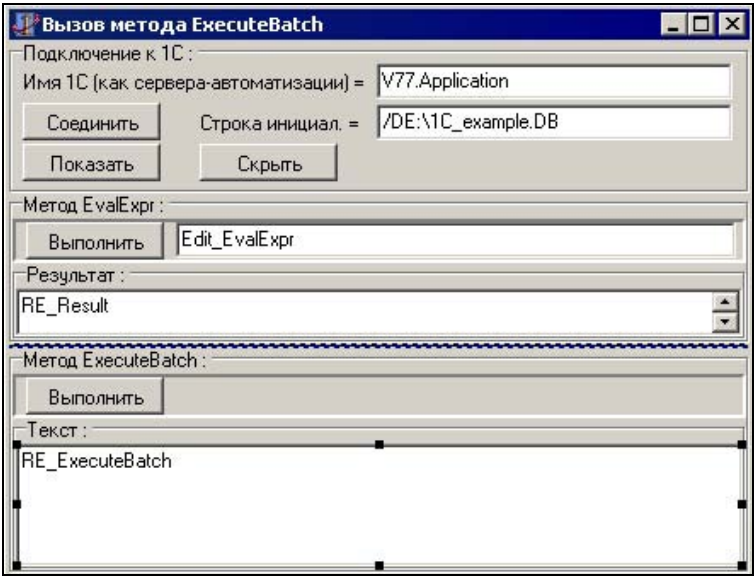



**Рис. 4.203.** Форма проекта sp_ExecuteBatch_1C.DPR

Как видите, здесь появились две новые кнопки **Показать** и **Скрыть** (группа **Подключение к 1С**), а также новая группа **Метод ExecuteBatch**.

Как уже говорилось, в некоторых случаях (например, вызов "штатной" функции `ОткрытьФорму`) необходимо, чтобы окно программы 1С:Предприятие (в режиме сервера OLE Automation) было открыто (по умолчанию в этом режиме оно скрыто). Рассмотрим процесс визуализации окна программы 1С:Предприяте более подробно.

В секцию `public` главной формы нашего Delphi-проекта внесем новое свойство `hW1C` (листинг 4.97).

**Листинг 4.97. Секция `public` главной формы Delphi-проекта**

```
public
  { Public declarations }
  // Ссылка на 1С, как сервер OLE Automation
  Ole_1C_App: OleVariant;
```

```
    // Дескриптор главного окна приложения 1С
    hW1C: hWND;
  end;
```

Для того чтобы открыть окно любого приложения в среде Windows, нужно получить дескриптор окна. Зная дескриптор окна приложения, можно вызвать функцию `ShowWindow` с соответствующими параметрами. Синтаксис:

```
BOOL ShowWindow(
    HWND hWnd,     // Дескриптор окна приложения
    int nCmdShow   // Режим визуализации окна приложения
   );
```

Из всех допустимых значений параметра `nCmdShow` в данном случае будем использовать два значения:

- `SW_SHOW` — показать окно приложения;
- `SW_HIDE` — скрыть окно приложения.

Остается только каким-то образом получить дескриптор окна запущенной программы 1С:Предприятие. Возможный способ — получить список всех окон приложений, находящихся в памяти компьютера, и выбрать то окно, заголовок которого совпадает с искомым. В листинге 4.98 представлен текст функции, формирующей список окон запущенных приложений.

**Листинг 4.98. Функция, формирующая список окон приложений, запущенных в среде Windows**

```
function sp_GetListOfWindowsNames(List: TStrings): integer;
// Получение списка окон запущенных приложений.
// Список окон формируется в переменной List (список).
// Возвращаемое значение — число строк в списке List.
Const
  Sep = '_|=>';
Var
 hW: hWND;
 Exit99: byte;
 Sx: string;
 CCC: array [0..255] of char;
begin
  Result:=0;
  if List<>NIL then begin
     List.Clear;
```

```
    hW:=GetTopWindow(0);
    if hW >32 then begin
       Exit99:=0;
       while Exit99<=0 do
        begin
         hW:=GetNextWindow(hW,GW_HWNDNEXT);
         if hW>32 then begin
            GetWindowText(hW,CCC,SizeOf(CCC));
            Sx:=StrPAS(CCC);
            Sx:=Trim(Sx);
            List.Add(Sx+Sep+IntToStr(hW));
         end
         else begin
            Exit99:=1;
         end;
       end;
    end;
    Result:=List.Count;
  end;
end;
```

Здесь для формирования списка окон и их дескрипторов используются системные функции Windows `GetTopWindow`, `GetNextWindow` и `GetWindowText`.

> ***Примечание***
>
> Функция `sp_GetListOfWindowsNames` не ориентирована исключительно на систему 1С, скорее, она имеет общесистемный характер. Поэтому подробно рассматривать ее мы не будем.

Заголовок окна программы 1С:Предприятие начинается с цепочки символов `"1С:Предприятие"`. То есть среди списка окон, сформированных функцией `sp_GetListOfWindowsNames`, нужно искать окно, заголовок которого начинается с этой цепочки символов. Эту операцию выполняет функция `sp_GetHandleThisWindow`, текст которой здесь не приводится по причине его тривиальности, но его можно найти в файле \sp_Lib\sp_Lib_Delphi.PAS на сопроводительном компакт-диске книги.

Изменим обработчик события "нажатие кнопки **Соединить**" следующим образом (листинг 4.99).

**Листинг 4.99. Обработчик события "нажатие кнопки *Соединить*"**

```
procedure TForm1.SpeedButton1Click(Sender: TObject);
// Соединение с 1С
begin
  Ole_1C_App:=UnAssigned;
  // Обнуляем значение переменной,
  // содержащей дескриптор окна 1С:Предприятие
  hW1C:=0;
  Edit_1Csrv.Color:=clWhite;
  Edit_1Cinit.Color:=clWhite;
  // Соединяемся с 1С
  Ole_1C_App:=ConnectTo1C(Edit_1Cinit.Text);
  if not VarIsEmpty(Ole_1C_App) then begin
     Edit_1Csrv.Color:=clYellow;
     Edit_1Cinit.Color:=clYellow;
     Edit_EvalExpr.SetFocus;
     //---------------------------------------------
     // Найдем окно запущенного приложения 1С и получим
     // дескриптор главного окна приложения 1С
     hW1C:=sp_GetHandleThisWindow('1С:Предприятие',1,TRUE,TRUE);
     if hW1C<=32 then begin
        hW1C:=0;
        ShowMessage('Главное окно приложения 1С не найдено !');
     end;
     //---------------------------------------------
  end;
end;
```

Поясним текст обработчика.

После того как программа 1С:Предприятие запущена в режиме сервера OLE Automation, производится поиск (функция `sp_GetHandleThisWindow`) окна программы, заголовок которой начинается с цепочки символов `"1С:Предприятие"`. Значение дескриптора найденного окна присваивается переменной `hW1C`.

Создадим обработчики события "нажатие кнопки" для кнопок **Показать** и **Скрыть** (листинги 4.100 и 4.101 соответственно).

**Листинг 4.100. Обработчик события "нажатие кнопки *Показать*"**

```
procedure TForm1.SpeedButton3Click(Sender: TObject);
begin
  if not VarIsEmpty(Ole_1C_App) then begin
     if hW1C>32 then begin
        // Показать главное окно приложения 1С
        ShowWindow(hW1C,SW_SHOW);
     end;
  end;
end;
```

**Листинг 4.101. Обработчик события "нажатие кнопки *Скрыть*"**

```
procedure TForm1.SpeedButton4Click(Sender: TObject);
begin
  if not VarIsEmpty(Ole_1C_App) then begin
     if hW1C>32 then begin
        // Скрыть главное окно приложения 1С
        ShowWindow(hW1C,SW_HIDE);
     end;
  end;
end;
```

В обоих случаях вызывается системная функция `ShowWindow` с соответствующим параметром.

***Примечание***

Исходные тексты проекта sp_ExecuteBatch_1C.DPR есть в папке \Part_1\Chapter_04\Topic_4_6\sp_ExecuteBatch_1C.PAS на сопроводительном компакт-диске книги.

Теперь можно проверить, как работает наш проект.

***Примечание***

Для поиска окна программы 1С:Предприятие мы использовали тот факт, что заголовок окна программы начинается с цепочки символов `"1С:Предприятие"`. Если в среде Windows запущено несколько приложений, окна которых начинаются с такой же цепочки символов, то обработчик `SpeedButton1Click` (листинг 4.99) сработает некорректно. На самом деле заголовок окна программы 1С:Предприятие имеет значительно больше символов и на практике для более точной идентификации окна программы нужно использовать этот факт.

В нашем случае просто закроем другие программы, заголовок окна которых начинается с цепочки символов "1С:Предприятие". Конфигуратор можно не закрывать, естественно.

Скомпилируем проект и запустим на выполнение. Нажмем кнопку **Соединить**, а затем кнопку **Показать**. Результат представлен на рис. 4.204.

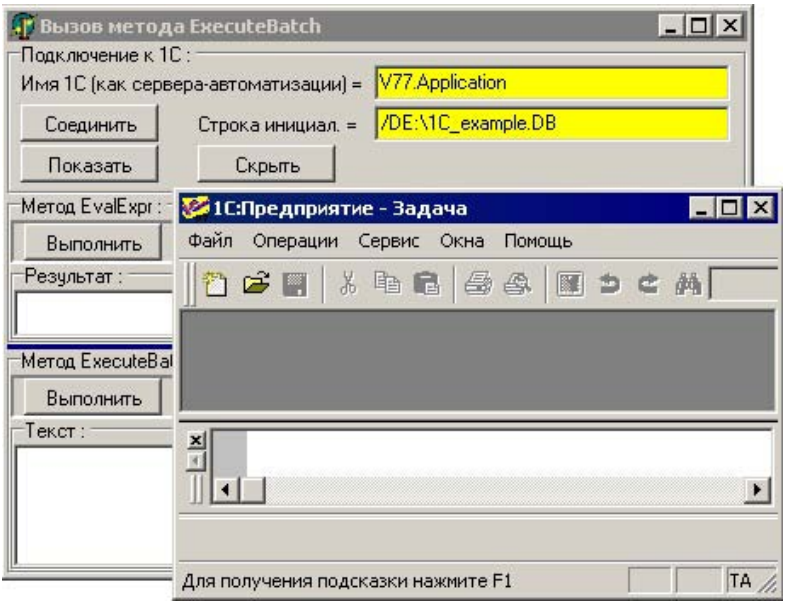



**Рис. 4.204.** Открытое окно программы 1С:Предприятие (в режиме сервера OLE Automation)

Здесь комментарии излишни.

По нажатию кнопки **Скрыть** окно программы 1С:Предпритяие исчезнет с экрана.

Закроем нашу программу и создадим обработчик события "нажатие кнопки **Выполнить**" (для `ExecuteBatch`). В листинге 4.102 представлен текст обработчика.

**Листинг 4.102. Обработчик события "нажатие кнопки *Выполнить*" (для `ExecuteBatch`)**

```
procedure TForm1.SpeedButton5Click(Sender: TObject);
begin
  if not VarIsEmpty(Ole_1C_App) then begin
```

```
      if hW1C>32 then begin
         // Показать главное окно приложения 1С
         ShowWindow(hW1C,SW_SHOW);
      end;
      //----------------------------------------
      // Вызываем метод ExecuteBatch
      if Ole_1C_App.ExecuteBatch(RE_ExecuteBatch.Text)*1>=0 then begin
         SbowMessage('Ошибка выполнения метода ExecuteBatch');
      end;
      //----------------------------------------
   end;
end;
```

Здесь пояснения не требуются.

Теперь можно проводить эксперименты. Для начала попробуем воспроизвести последовательность команд, выполненных в предыдущем разделе с помощью метода `EvalExpr` (листинг 4.93). Для этого введем те же команды в поле `Текст` для `ExecuteBatch` (в нашей программе) и в конце добавим строку:

```
Сообщить(гл9БуферДляOLE[3]);
```

### *Примечание*

В отличие от метода `EvalExpr`, в методе `ExecuteBatch` каждую команду нужно отделять точкой с запятой. То есть нужно абсолютно точно придерживаться синтаксиса встроенного языка программирования системы 1С:Предприятие.

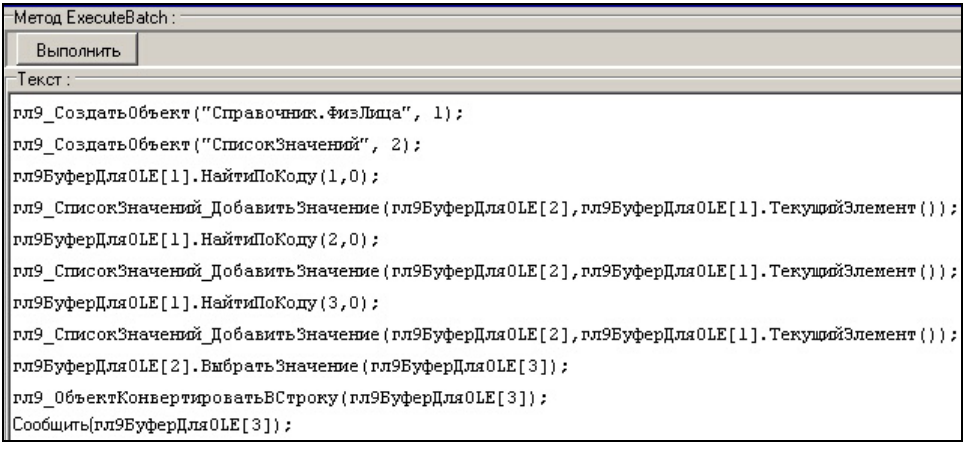



**Рис. 4.205.** Текст последовательности команд (листинг 4.93) для их выполнения с помощью метода `ExecuteBatch`

Запустим этот набор команд на выполнение (кнопка **Выполнить**). Если все сделано без ошибок, то на экране появится диалоговое окно для выбора строки из списка значений (рис. 4.206).

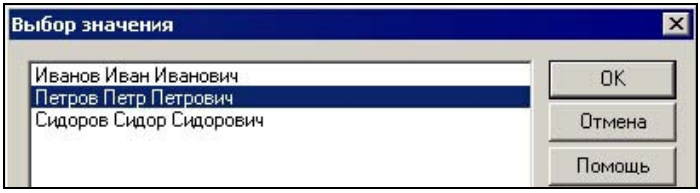



**Рис. 4.206.** Результат выполнения команды выбора строки из списка значений

Это "сработала" строка:

```
гл9БуферДляOLE[2].ВыбратьЗначение(гл9БуферДляOLE[3]);
```

В окне сообщений программы 1С:Предприятие (рис. 4.207) появится строка с ФИО выбранного "физического лица".

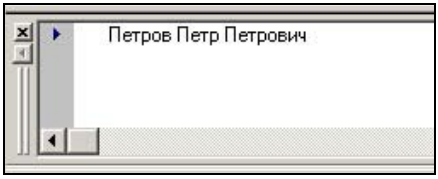



**Рис. 4.207.** Окно сообщений программы 1С:Предприятие

Результат можно сравнить с получаемым при поочередном выполнении этих же команд с помощью метода `EvalExpr`.

Теперь посмотрим, каким образом мы можем использовать управляющие конструкции и операторы в тексте, выполняемом методом `ExecuteBatch`. Поставим перед собой следующую задачу:

1. Создать объект `СписокЗначений` и разместить в нем ссылки на все элементы справочника `ФизЛица`, кроме групп.
2. Из списка значений вывести в окно сообщений программы 1С:Предприятие информацию о коде, ФИО и городе проживания отобранных "физических лиц".

В листинге 4.103 представлен текст для метода `ExecuteBatch`, решающий эту задачу.

**Листинг 4.103. Отбор всех "физических лиц" в список значений и вывод информации о них в окно сообщений программы 1С:Предприятие**

```
ОчиститьОкноСообщений();
Сообщить("Список физ. лиц");
Сообщить("==================");
//----------------------------------------
// Создаем список значений.
// Ссылка на список значений будет храниться
// в элементе массива гл9БуферДляOLE[2].
гл9_СоздатьОбъект("СписокЗначений", 2);
//----------------------------------------
//----------------------------------------
// Создаем объект "Справочник.ФизЛица".
// Ссылка на справочник будет храниться
// в элементе массива гл9БуферДляOLE[1].
гл9_СоздатьОбъект("Справочник.ФизЛица", 1);
//----------------------------------------
Сообщить("Выбираем из справочника:");
//----------------------------------------
// Выбираем из справочника всех "физических лиц"
// так же, как мы это делали ранее
гл9БуферДляOLE[1].ВыбратьЭлементы(1);
Пока гл9БуферДляOLE[1].ПолучитьЭлемент()>0 Цикл
   Если гл9БуферДляOLE[1].ЭтоГруппа()<=0 Тогда
     // Ссылка на текущий элемент справочника
     // будет храниться в элементе массива гл9БуферДляOLE[3]
     гл9БуферДляOLE[3] = гл9БуферДляOLE[1].ТекущийЭлемент();
     // Выводим сообщение в окно сообщений 1С
     Сообщить(СокрЛП(гл9БуферДляOLE[3].Код)+"  "
                                +СокрЛП(гл9БуферДляOLE[3]));
     // Сохраняем ссылку на текущий элемент справочника
     // в списке значений
     гл9_СписокЗначений_ДобавитьЗначение(гл9БуферДляOLE[2],
                                         гл9БуферДляOLE[3]);
   КонецЕсли;
КонецЦикла;
//----------------------------------------
```

```
Сообщить("===================");
Сообщить("");
Сообщить("Выбираем из списка значений:");
//----------------------------------------
// Теперь сканируем весь список значений
// и поочередно выбираем из него
// ссылки на текущий элемент справочника.
// Определяем количество строк в списке значений
гл9БуферДляOLE[10] = гл9БуферДляOLE[2].РазмерСписка();
// Это будет "счетчик циклов"
гл9БуферДляOLE[11] = 0;
Пока гл9БуферДляOLE[11]<гл9БуферДляOLE[10] Цикл
  // Увеличиваем значение счетчика циклов на 1
  гл9БуферДляOLE[11]=гл9БуферДляOLE[11]+1;
  // Считываем из списка значений ссылку
  // на текущий элемент справочника
  гл9БуферДляOLE[12] =
          гл9БуферДляOLE[2].ПолучитьЗначение(гл9БуферДляOLE[11]);
  // Выводим в окно сообщений 1С
  Сообщить(СокрЛП(гл9БуферДляOLE[12].Код)
          +" "
          +СокрЛП(гл9БуферДляOLE[12])
          +" ("
          +СокрЛП(гл9БуферДляOLE[12].ГородПроживания)
          +")");
КонецЦикла;
//----------------------------------------
Сообщить("===================");
```

Поясним текст листинга 4.103. В принципе, здесь нет ничего такого, чего мы не делали ранее, кроме, разве что, функции `ОчиститьОкноСообщений` и метода `ПолучитьЗначение` объекта `СписокЗначений`. Непривычным, возможно, является использование элемента массива `гл9БуферДляOLE` в качестве счетчика циклов (элемент массива `гл9БуферДляOLE[11]`). В остальном этот текст ничем не отличается от того, что рассматривалось в *разд. 4.1* и *4.5*, потому углубляться в него не будем.

Результат выполнения этого кода в методе `ExecuteBatch` представлен на рис. 4.208.

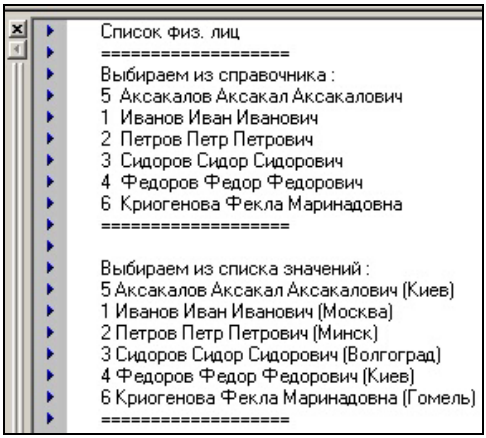



**Рис. 4.208**. Результат выполнения кода из листинга 4.103

Программный текст, представленный в листинге 4.103, полезен тем, что показывает, каким образом можно реализовать довольно сложные алгоритмы получения информации из базы данных 1С:Предприятие (используя механизм OLE Automation), не создавая при этом дополнительных программ.

С практической точки зрения, вывода информации в окно сообщений программы 1С:Предприятие может оказаться недостаточно. В листинге 4.104 представлен программный текст, который "делает" то же самое, что и в предыдущем примере, но сохраняет результат в текстовом файле.

**Листинг 4.104. Отбор всех "физических лиц" и сохранение полученного списка в текстовом файле на диске**

```
ОчиститьОкноСообщений();
// Имя начального каталога
гл9БуферДляOLE[51] = "C:\";
// Имя файла
гл9БуферДляOLE[50] = "1C_ExecuteBatch_0001.txt";
// Выбираем файл для сохранения списка "физических лиц"
Если ФС.ВыбратьФайл(1,
                    гл9БуферДляOLE[50],
                    гл9БуферДляOLE[51],
                    "Сохранить в файл",
                    "Текстовые файлы ( *.txt )!*.txt", , ) = 1 Тогда
 Сообщить("Результат будет сохранен в файле: "
                +гл9БуферДляOLE[51]+гл9БуферДляOLE[50]);
```

```
Сообщить("Список физ. лиц");
Сообщить("==================");
//----------------------------------------
// Создаем объект типа "Текст",
// ссылка на который будет храниться
// в элементе массива гл9БуферДляOLE[2]
гл9_СоздатьОбъект("Текст", 2);
//----------------------------------------
//----------------------------------------
// Создаем объект "Справочник.ФизЛица",
// ссылка на этот справочник будет храниться
// в элементе массива гл9БуферДляOLE[1]
гл9_СоздатьОбъект("Справочник.ФизЛица", 1);
//----------------------------------------
Сообщить("Выбираем из справочника:");
//----------------------------------------
// Выбираем из справочника всех "физических лиц"
// так же, как мы это делали раньше
гл9БуферДляOLE[1].ВыбратьЭлементы(1);
Пока гл9БуферДляOLE[1].ПолучитьЭлемент()>0 Цикл
 Если гл9БуферДляOLE[1].ЭтоГруппа()<=0 Тогда
  // Ссылка на текущий элемент справочника
  // будет храниться в элементе массива гл9БуферДляOLE[3]
  гл9БуферДляOLE[3] = гл9БуферДляOLE[1].ТекущийЭлемент();
  // Формируем строку для вывода
  гл9БуферДляOLE[5] = СокрЛП(гл9БуферДляOLE[3].Код)
                          +"  "
                          +СокрЛП(гл9БуферДляOLE[3]);
  // Выводим строку в окно сообщений 1С
  Сообщить(гл9БуферДляOLE[5]);
  // Добавляем строку в объект типа "Текст", ссылка на который
  // хранится в элементе массива гл9БуферДляOLE[2]
  гл9БуферДляOLE[2].ДобавитьСтроку(гл9БуферДляOLE[5]);
 КонецЕсли;
КонецЦикла;
//----------------------------------------
Сообщить("==================");
// Сохраняем содержимое объекта гл9БуферДляOLE[2] типа "Текст"
```

```
// в файле на диске
 гл9БуферДляOLE[2].Записать(гл9БуферДляOLE[51]+гл9БуферДляOLE[50]);
 // Очищаем содержимое объекта гл9БуферДляOLE[2]
 гл9БуферДляOLE[2].Очистить();
 // Прочитаем сохраненную информацию из файла с помощью
// метода "Открыть" объекта гл9БуферДляOLE[2]
 гл9БуферДляOLE[2].Открыть(гл9БуферДляOLE[51]+гл9БуферДляOLE[50]);
 // Устанавливаем режим "Только просмотр"
 гл9БуферДляOLE[2].ТолькоПросмотр(1);
 // Отображаем содержимое файла в диалоговом окне
 гл9БуферДляOLE[2].Показать("Список физ.лиц");
КонецЕсли;
```

Поясним текст листинга 4.104. Новым здесь является использование объектов `ФС` (файловая система) и `Текст`. Прежде всего, в результате вызова метода `ВыбратьФайл` объекта `ФС`:

```
Если ФС.ВыбратьФайл(1,
```

на экране появляется диалоговое окно выбора имени файла для (рис. 4.209).

**Рис. 4.209**. Окно выбора файла (метод `ВыбратьФайл` объекта `ФС`)

Далее создается объект типа `Текст`:

```
гл9_СоздатьОбъект("Текст", 2);
```

а также объект `Справочник.ФизЛица`.

Потом следует отбор всех элементов справочника, соответствующих "физическим лицам". Для каждого "физического лица" формируется соответствующая строка:

```
гл9БуферДляOLE[5] = СокрЛП(гл9БуферДляOLE[3].Код)
                                    +"  "+СокрЛП(гл9БуферДляOLE[3]);
```

***Примечание***

В отличие от объектов `СписокЗначений` и `ТаблицаЗначений`, объект `Текст` не может хранить значения никаких иных типов данных, кроме строк.

Затем сформированная строка записывается в объект типа `Текст`:

```
гл9БуферДляOLE[2].ДобавитьСтроку(гл9БуферДляOLE[5]);
```

и выводится в окно сообщений программы 1С:Предприятие (рис. 4.210).

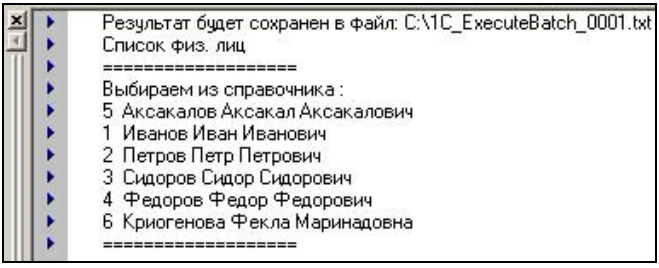



**Рис. 4.210.** Результат выполнения кода из листинга 4.104

По окончании считывания всех элементов справочника из выборки содержимое объекта `Текст` сохраняется в указанном файле:

```
гл9БуферДляOLE[2].Записать(гл9БуферДляOLE[51]+гл9БуферДляOLE[50]);
```

Здесь: `гл9БуферДляOLE[51]` — имя каталога (папки); `гл9БуферДляOLE[50]` — имя файла.

Содержание объекта типа `Текст` (для чистоты эксперимента) после этого очищается:

```
гл9БуферДляOLE[2].Очистить();
```

Информация, сохраненная в файле, вновь считывается в объект типа `Текст`:

```
гл9БуферДляOLE[2].Открыть(гл9БуферДляOLE[51]+гл9БуферДляOLE[50]);
```

Устанавливаем режим `ТолькоПросмотр` для объекта типа `Текст`:

```
гл9БуферДляOLE[2].ТолькоПросмотр(1);
```

И содержание выводится на экран в диалоговом окне (рис. 4.211):

```
гл9БуферДляOLE[2].Показать("Список физ.лиц");
```

**Рис. 4.211.** Результат выполнения метода `Показать` объекта типа `Текст`

Вот, собственно, и все.

Существенный интерес представляет возможность формирования и выполнения "штатных" запросов системы 1С:Предприятие к базе данных.

***Определение***

*Запрос* — это средство для выполнения обращений к документам, регистрам, справочникам и журналам расчетов с целью получения сводной информации при формировании выходных отчетов.

С помощью "запросов" можно фильтровать информацию по различным критериям, создавать группировки отфильтрованной информации, получать сводную информацию. "Запросы" можно использовать не только в рамках отчетов, но и как самостоятельные объекты. Если учесть, что в системе 1С:Предприятие есть "штатное" средство для построения отчетов, в рамках которого автоматизирован процесс построения запросов (их тексты можно использовать впоследствии), то ясно, насколько полезным может быть использование "запросов" при доступе к информации базы данных системы 1С:Предприятие.

***Примечание***

Объект `Запрос`, его методы и свойства являются достаточно сложными и "объемными", чтобы рассматривать их в рамках данной книги. Подробную информацию о "запросах" можно найти в специальной литературы по системе 1С:Предприятие.

Здесь мы рассмотрим использование запроса для выборки информации по регистру `ПартииНаличие` стандартной конфигурации 1С:Предприятие. Это связано с тем, что (как правило) на всех предприятиях имеется существенное количество записей движения по регистру `ПартииНаличие`, поэтому выполнение запроса по этому регистру будет хорошей иллюстрацией к рассматриваемой теме.

В листинге 4.105 представлен пример формирования и выполнения запроса по регистру `ПартииНаличие` стандартной конфигурации 1С:Предприятие с использованием метода `ExecuteBatch`.

**Листинг 4.105. Использование запросов для доступа к информации с помощью метода `ExecuteBatch`**

```
ОчиститьОкноСообщений();
Сообщить("Начало");
//=====================================
//Задаем период дат для отбора движений
гл9БуферДляOLE[5] = "01.01.2003";
гл9БуферДляOLE[6] = "31.01.2003";
Сообщить("Период:"+СокрЛП(гл9БуферДляOLE[5])
                      +" "+СокрЛП(гл9БуферДляOLE[6]));
//=====================================
//=====================================
//Формируем Запрос
//Создать объект Запрос
гл9БуферДляOLE[1] = СоздатьОбъект("Запрос");
// Текст запроса  (гл9БуферДляOLE[2])
гл9БуферДляOLE[2] =
  "//{{ЗАПРОС(Сформировать)
  |Период с '"+гл9БуферДляOLE[5] + "' по '"+ гл9БуферДляOLE[6]+"';
  |МОЛ = Регистр.ПартииНаличие.МОЛ;
  |Номенклатура = Регистр.ПартииНаличие.Номенклатура;
  |Партия = Регистр.ПартииНаличие.Партия;
  |ДатаПартии = Регистр.ПартииНаличие.ДатаПартии;
  |Количество = Регистр.ПартииНаличие.Количество;
  |СуммаРуб = Регистр.ПартииНаличие.СуммаРуб;
  |Функция КоличествоПриход = Приход(Количество);
  |Функция СуммаРубПриход = Приход(СуммаРуб);
  |Функция КоличествоРасход = Расход(Количество);
  |Функция СуммаРубРасход = Расход(СуммаРуб);
  |Группировка МОЛ без групп;
  |Группировка Номенклатура без групп;
  |"//}}ЗАПРОС
;
// Для контроля выводим текст запроса
```

```
// в окно сообщений 1С
Сообщить("Текст Запроса: ");
Сообщить(гл9БуферДляOLE[2]);
//=========================================
//=========================================
// Запускаем запрос на выполнение
Если гл9БуферДляOLE[1].Выполнить(гл9БуферДляOLE[2]) = 0 Тогда
   // Возврат; // С этим не работаем
КонецЕсли;
//=========================================
//=========================================
// Создаем таблицу значений
гл9БуферДляOLE[3]=СоздатьОбъект("ТаблицаЗначений");
// Выгружаем Запрос в таблицу значений
гл9БуферДляOLE[1].Выгрузить(гл9БуферДляOLE[3]);
// Для контроля отображаем в диалоговом окне
гл9БуферДляOLE[3].ВыбратьСтроку();
//=========================================
```

Поясним листинг 4.105. Сначала создается объект типа `Запрос`:

```
гл9БуферДляOLE[1] = СоздатьОбъект("Запрос");
```

Затем формируется текст запроса (переменная `гл9БуферДляOLE[2]`). Фильтрация производится по диапазону дат (переменные `гл9БуферДляOLE[5]` и `гл9БуферДляOLE[6]`). Информация выбирается по ресурсам `Количество` и `СуммаРуб` регистра и группируется по измерениям `Номенклатура` и `МОЛ` регистра.

Далее (для контроля), в строке:

```
Сообщить(гл9БуферДляOLE[2]);
```

текст запроса выводится в окно сообщений (рис. 4.212).

Наконец, вызывается метод `Выполнить` объекта `Запрос`:

```
Если гл9БуферДляOLE[1].Выполнить(гл9БуферДляOLE[2]) = 0 Тогда
```

в результате которого формируется соответствующим образом отфильтрованная и сгруппированная информация.

Затем создается объект `ТаблицаЗначений`:

```
гл9БуферДляOLE[3]=СоздатьОбъект("ТаблицаЗначений");
```

куда выгружается результат выполнения запроса (с помощью метода `Выгрузить` объекта `Запрос`):

```
гл9БуферДляOLE[1].Выгрузить(гл9БуферДляOLE[3]);
```

```
Начало
Период:  01.01.2003  31.01.2003
Текст Запроса :
//{{ЗАПРОС(Сформировать)
Период с '01.01.2003' по '31.01.2003';
МОЛ = Регистр.ПартииНаличие.МОЛ;
Номенклатура = Регистр.ПартииНаличие.Номенклатура;
Партия = Регистр.ПартииНаличие.Партия;
ДатаПартии = Регистр.ПартииНаличие.ДатаПартии;
Количество = Регистр.ПартииНаличие.Количество;
СуммаРуб = Регистр.ПартииНаличие.СуммаРуб;
Функция КоличествоПриход = Приход(Количество);
Функция СуммаРубПриход = Приход(СуммаРуб);
Функция КоличествоРасход = Расход(Количество);
Функция СуммаРубРасход = Расход(СуммаРуб);
Группировка МОЛ без групп;
Группировка Номенклатура без групп;
```

**Рис. 4.212.** Текст запроса, выведенный в окно сообщений 1С:Предприятие

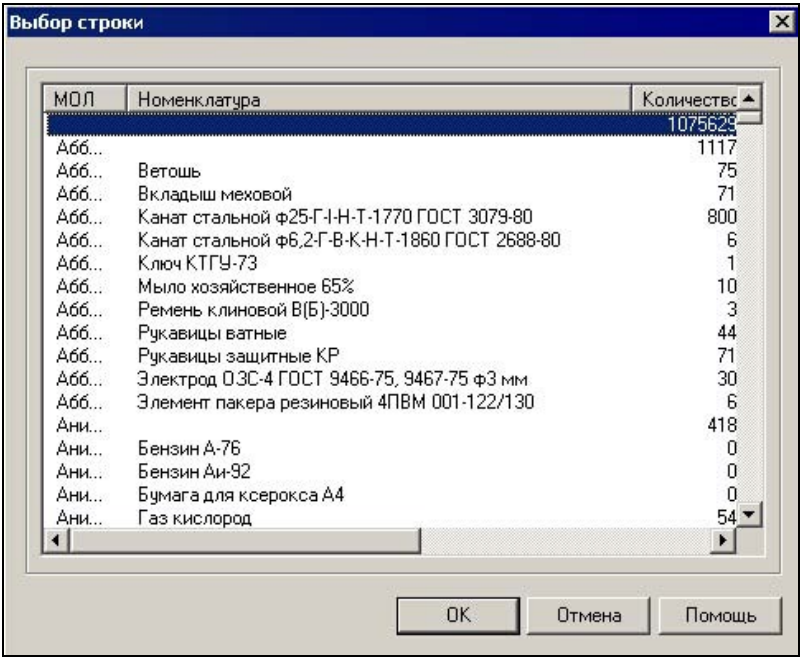



**Рис. 4.213.** Результат вызова метода `ВыбратьСтроку` объекта `ТаблицаЗначений`

После окончания выгрузки содержимое объекта `ТаблицаЗначений` можно посмотреть, обратившись к его методу `ВыбратьСтроку`:

```
гл9БуферДляOLE[3].ВыбратьСтроку();
```

На рис. 4.213 представлено диалоговое окно (как результат вызова метода `ВыбратьСтроку` объекта `ТаблицаЗначений`).

***Примечание***

Можно достаточно легко (с помощью встроенного языка программирования системы 1С:Предприятие) написать универсальную функцию выгрузки содержимого объекта `ТаблицаЗначений` в текстовый файл, в DBF-файл или в файл Excel. Если поместить эту функцию в глобальный модуль конфигурации с ключевым словом `Экспорт`, то можно во внешнем приложении сформировать запрос, выполнить его (используя метод `ExecuteBatch`) и выгрузить результат запроса сначала в таблицу значений, а потом, например, в файл Excel или в DBF-файл. И далее воспользоваться средствами Delphi для работы с этим DBF-файлом по своему усмотрению.

Вернемся к методу `EvalExpr`, который был подробно рассмотрен в предыдущем разделе. Как было отмечено, метод `EvalExpr` можно использовать для "запуска" отчетов и других внешних форм (обработок). Но чтобы открытая форма была доступна пользователю, нужно открыть главное окно программы 1С:Предприятие. В этом разделе мы можем вернуться к этому вопросу.

Откроем "нашу" конфигурацию и создадим внешнюю обработку. На рис. 4.214 представлена форма обработки.

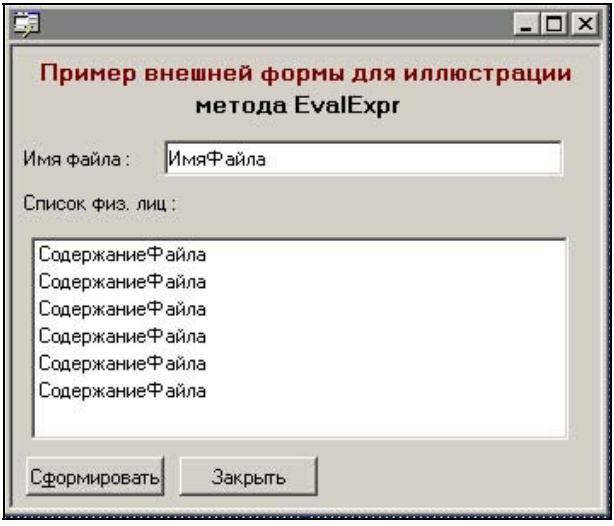



**Рис. 4.214**. Форма внешней обработки для просмотра текстовых файлов.

Здесь `СодержаниеФайла` — это элемент управления типа "список", `ИмяФайла` — это элемент управления типа "однострочный редактор").

Форма может получить имя текстового файла при запуске, через параметр формы.

По нажатию кнопки **Сформировать** проверяется, существует ли заданный файл. И если существует, то считывается его значение.

В листинге 4.106 представлен текст обработчика события "нажатие кнопки **Сформировать**" (рис. 4.214).

**Листинг 4.106. Обработчик события "нажатие кнопки *Сформировать*"**

```
Процедура Сформировать()
  // Очищаем элемент управления "СодержаниеФайла"
  СодержаниеФайла.УдалитьВсе();
  // Проверяем, существует ли заданный файл
  Если ФС.СуществуетФайл(ИмяФайла)>0 Тогда
    //-----------------------------
    // Считываем содержание файла с именем,
    // хранящимся в элементе управления "ИмяФайла"
    ТекстБуф = СоздатьОбъект("Текст");
    ТекстБуф.Открыть(ИмяФайла);
    //-----------------------------
    //-----------------------------
    // Копируем прочитанное содержание файла
    // из объекта "ТекстБуф" в элемент управления
    // "СодержаниеФайла"
    Колво = ТекстБуф.КоличествоСтрок();
    i = 0;
    Пока i<Колво Цикл
      i = i + 1;
      Sx = ТекстБуф.ПолучитьСтроку(i);
      СодержаниеФайла.ДобавитьЗначение(Sx,Sx);
    КонецЦикла;
    //-----------------------------
  Иначе
    Предупреждение("Файл: <"+ИмяФайла+"> не найден");
  КонецЕсли;
КонецПроцедуры
```

Вкратце поясним текст обработчика (листинг 4.106).

Как было сказано выше, сначала проверяется существование заданного файла (метод `СуществуетФайл` объекта `ФС`). Далее создается объект типа `Текст`,

куда считывается текстовый файл (метод `Открыть` объекта `ТекстБуф`). Затем информация из объекта `ТекстБуф` копируется в элемент управления `СодержаниеФайла`.

При интерактивном открытии формы система 1С:Предприятие автоматически вызывает предопределенную процедуру `ПриОткрытии`, если она имеется в тексте модуля данной формы. При этом, если при "запуске" данной форме был передан параметр, он может быть прочитан и проанализирован в процедуре `ПриОткрытии` с помощью свойства `Форма.Параметр`.

В листинге 4.107 представлен текст предопределенной процедуры `ПриОткрытии` для созданной формы.

**Листинг 4.107. Обработчик события "при открытии формы"**

```
Процедура ПриОткрытии()
  Yes=0;
  // Получаем параметр, переданный форме
  Парам = Форма.Параметр;
  // Очищаем элемент управления "ИмяФайла"
  ИмяФайла = "";
  // Очищаем элемент управления "СодержаниеФайла"
  СодержаниеФайла.УдалитьВсе();
  // Проверяем, соответствует ли тип переданного
  // параметра типу "Строка"
  Если ТипЗначения(Парам)=2 Тогда
    // Тип переданного значения — "Строка"
    Парам = СокрЛП(Парам);
    Если СтрДлина(Парам) >0 Тогда
      // Имя файла передано
      ИмяФайла = Парам;
      Yes=1;
    КонецЕсли;
  КонецЕсли;
  Если Yes>0 Тогда
    // Если все ОК — считываем содержание
    // файла (имя передано в параметре формы)
    Сформировать();
  Иначе
    // Иначе — сообщение об ошибке
    Предупреждение("Ошибка при инициализации отчета(формы)");
```

```
  КонецЕсли;
  // СтатусВозврата(0);
  // Возврат;
КонецПроцедуры
```

Кратко поясним текст обработчика (листинг 4.107). Здесь проверяется тип значения переданного параметра. Если тип значения — `Строка`, и она не пуста, то ее значение присваивается элементу управления `ИмяФайла` и вызывается обработчик события "нажатие кнопки **Сформировать**".

> ***Примечание***
>
> Файл этой формы на сопроводительном компакт-диске книги: \1C_example.DB\ExtForms\ВнешняяФорма_для_EvalExpr.ERT.

Теперь запустим наш проект (sp_ExecuteBatch_1C.DPR) на выполнение, создадим соединение с 1С и сделаем окно программы 1С видимым, а затем введем в поле значения параметра для `EvalExpr` строку:

```
ОткрытьФорму("Отчет#", "C:\1C_ExecuteBatch_0001.txt",
            "E:\1C_example.DB\ExtForms\ВнешняяФорма_для_EvalExpr.ert")
```

и нажмем кнопку **Выполнить**.

Результат представлен на рис. 4.215.

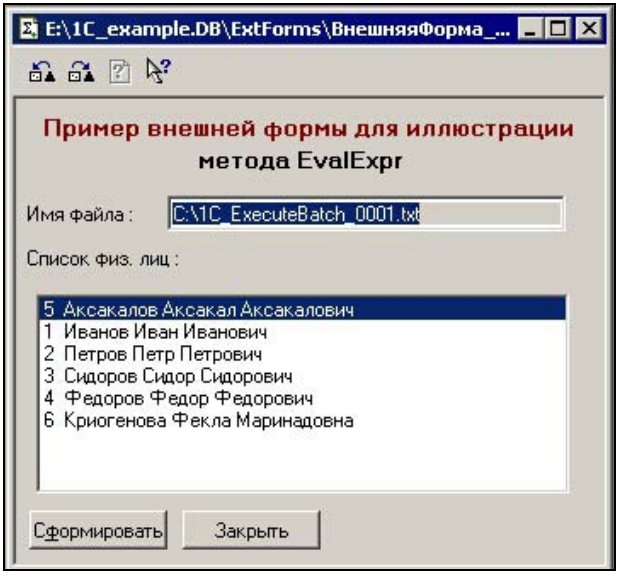



**Рис. 4.215.** Просмотр содержания текстового файла во внешней обработке

То есть было прочитано содержание текстового файла, сформированного в процессе выполнения программного кода (см. листинг 4.104).

> ***Примечание***
>
> Естественно, предполагается, что файлы C:\1C_ExecuteBatch_0001.TXT и E:\1C_example.DB\ExtForms\ВнешняяФорма_для_EvalExpr.ert существуют.

Вместо указания полного пути к файлу вызываемой формы можно использовать функцию `КаталогИБ()` (ее мы рассматривали в *разд. 4.5*):

```
ОткрытьФорму("Отчет#", "C:\1C_ExecuteBatch_0001.txt",
            КаталогИБ()+"ExtForms\ВнешняяФорма_для_EvalExpr.ert")
```

# 4.7. Изменение информации в базе данных 1С

В предыдущих разделах мы коснулись вопросов считывания информации из базы данных системы 1С:Предприятие с использованием механизма OLE Automation. Чаще всего именно эти действия бывает нужно выполнить при обмене данными с системой 1С:Предприятие. Тем не менее, иной раз возникает необходимость редактировать базу данных 1С:Предприятие. В этом разделе мы рассмотрим вопросы записи информации в базу данных на конкретных примерах.

## 4.7.1. Редактирование элементов справочников

Для начала рассмотрим редактирование непериодических реквизитов справочника.

Запустим Конфигуратор 1С:Предприятие и внесем новый (непериодический) реквизит `СостоитВПрофсоюзе` типа `Число` (рис. 4.216).

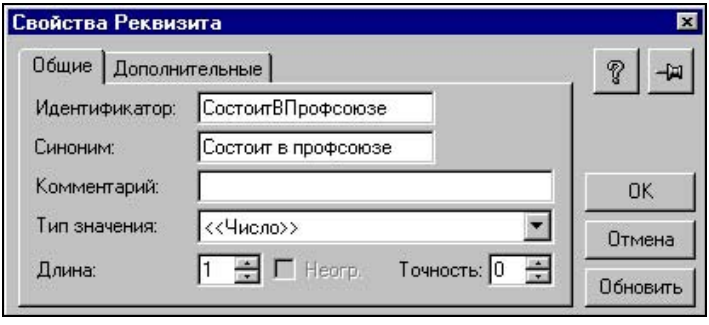



**Рис. 4.216.** Новый реквизит типа `Число` в справочнике `ФизЛица`

**Рис. 4.217.** Список реквизитов справочника `ФизЛица`

На рис. 4.147 представлен список реквизитов справочника `ФизЛица`.

Нужно внести также соответствующие изменения в форму редактирования элемента справочника `ФизЛица`, как мы это делали в *разд. 4.1.2* (см. рис. 4.16).

Сохраним изменения в конфигурации, запустим программу 1С:Предприятие и внесем значения реквизита `СостоитВПрофсоюзе` в справочник `ФизЛица` так, как указано в табл. 4.11.

***Таблица 4.11.*** *Справочник `ФизЛица`. Значение реквизита `СостоитВПрофсоюзе`*

| Код | ФИО | Значение признака |
|---|---|---|
| 5 | Аксакалов Аксакал Аксакалович | 0 |
| 1 | Иванов Иван Иванович | 1 |
| 6 | Криогенова Фекла Маринадовна | 0 |
| 2 | Петров Петр Петрович | 1 |
| 3 | Сидоров Сидор Сидорович | 0 |
| 4 | Федоров Федор Федорович | 1 |

Откроем Delphi и создадим новый проект с именем (например) sp_EditAttrInRef.DPR.

***Примечание***

Исходники этого примера (проект sp_EditAttrInRef.DPR) находтся на CD в папке \Part_1\Chapter_04\Topic_4_7\Item_4_7_1\sp_EditAttrInRef.PAS\01.

Форма проекта представлена на рис. 4.218.

В табл. 4.12 приведено назначение элементов управления на форме проекта.

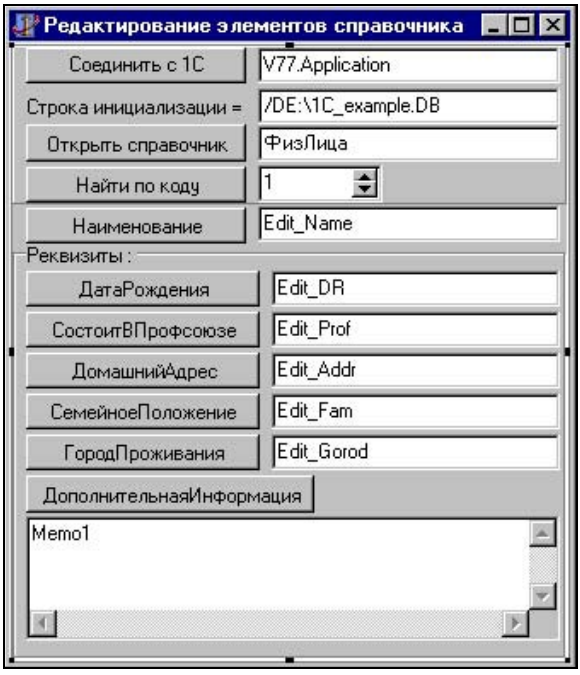



**Рис. 4.218.** Форма проекта sp_EditAttrInRef.DPR

***Таблица 4.12.*** *Назначение элементов управления в форме проекта sp_EditAttrInRef.DPR*

| **Элемент управления** | **Идентификатор** | **Назначение** |
|---|---|---|
| Кнопка **Соединить с 1С** | `Button1: TButton` | Соединение с программой 1С:Предприятие |
| Кнопка **Открыть справочник** | `SpeedButton1: TSpeedButton` | Открытие справочника `ФизЛица` |
| Кнопка **Найти по коду** | `SpeedButton9: TspeedButton` | Вызов метода `НайтиПоКоду` справочника `ФизЛица` для поиска элемента с кодом `SE_Code.Value` |
| Счетчик (поле ввода с кнопками) | `SE_Code: TspinEdit` | Код элемента справочника |
| Кнопка **Наименование** | `SpeedButton7: TspeedButton` | Редактирование свойства `Наименование` элемента справочника |

***Таблица 4.12*** *(окончание)*

| **Элемент управления** | **Идентификатор** | **Назначение** |
|---|---|---|
| Однострочный редактор | `Edit_Name: TEdit` | Значение свойства `Наименование` элемента справочника |
| Кнопка **ДатаРождения** | `SpeedButton2: TSpeedButton` | Редактирование реквизита `ДатаРождения` элемента справочника |
| Однострочный редактор | `Edit_DR: TEdit` | Значение реквизита `ДатаРождения` элемента справочника |
| Кнопка **СостоитВПрофсоюзе** | `SpeedButton3: TSpeedButton` | Редактирование реквизита `СостоитВПрофсоюзе` элемента справочника |
| Однострочный редактор | `Edit_Prof: TEdit` | Значение реквизита `СостоитВПрофсоюзе` элемента справочника |
| Кнопка **ДомашнийАдрес** | `SpeedButton4: TSpeedButton` | Редактирование реквизита `ДомашнийАдрес` элемента справочника |
| Однострочный редактор | `Edit_Addr: TEdit` | Значение реквизита `ДомашнийАдрес` элемента справочника |
| Кнопка **СемейноеПоложение** | `SpeedButton5: TSpeedButton` | Редактирование реквизита `СемейноеПоложение` элемента справочника |
| Однострочный редактор | `Edit_Fam: TEdit` | Значение реквизита `СемейноеПоложение` элемента справочника |
| Кнопка **ГородПроживания** | `SpeedButton6: TSpeedButton` | Редактирование реквизита `ГородПроживания` элемента справочника |
| Однострочный редактор | `Edit_Gorod: TEdit` | Значение реквизита `ГородПроживания` элемента справочника |
| Кнопка **ДополнительнаяИнформация** | `SpeedButton8: TSpeedButton` | Редактирование реквизита `ДополнительнаяИнформация` элемента справочника |
| Многострочный редактор | `Memo1: TMemo` | Значение реквизита `ДополнительнаяИнформация` элемента справочника |

***Примечание***

Вопросы соединения с программой 1С:Предприятие, открытия справочников и позиционирования на элементах справочника были подробно рассмотрены в предыдущих разделах.

В листинге 4.108 представлен исходный текст обработчика события "нажатие кнопки **Найти по коду**".

**Листинг 4.108. Визуализация элемента справочника ФизЛица (непериодические реквизиты)**

```
procedure TForm1. SpeedButton9Click(Sender: TObject);
// Обработчик события "нажатие кнопки "НайтиПоКоду"
// Визуализация элемента справочника
begin
 Edit_Name.Text:='???';
 Edit_DR.Text:='???';
 Edit_Prof.Text:='???';
 Edit_Addr.Text:='???';
 Edit_Fam.Text:='???';
 Edit_Gorod.Text:='???';
 Memo1.Lines.Clear;
 GroupBox_Attr.Visible:=TRUE;
 // Проверяем, существует ли ссылка на справочник
 if not VarIsEmpty(ole_Ref) then begin
   // Позиционируем справочник на элементе с заданным кодом
   if (ole_Ref.НайтиПоКоду(SE_Code.Value,0)*1)>0 then begin
     // Проверяем, является ли элемент справочника группой
     if (ole_Ref.ЭтоГруппа*1)<=0 then begin
       // Это — не группа (показываем реквизиты)
       GroupBox_Attr.Visible:=TRUE;
       //-------------------------------------
       // Читаем информацию из справочника
       Edit_Name.Text:=Trim(ole_Ref.Наименование);
       Edit_DR.Text:=Trim(ole_Ref.ДатаРождения);
       Edit_Prof.Text:=Trim(ole_Ref.СостоитВПрофсоюзе);
       Edit_Addr.Text:=Trim(ole_Ref.ДомашнийАдрес);
       Edit_Fam.Text:=Trim(ole_Ref.СемейноеПоложение.Идентификатор);
       Edit_Gorod.Text:=Trim(ole_Ref.ГородПроживания.Наименование);
       Memo1.Text:=Trim(ole_Ref.ДополнительнаяИнформация);
       //-------------------------------------
```

```
      end
      else begin
        // Это — группа (скрываем реквизиты)
        GroupBox_Attr.Visible:=FALSE;
        // Читаем из справочника только наименование
        Edit_Name.Text:=Trim(ole_Ref.Наименование);
      end;
    end
    else begin
       ShowMessage('Элемент справочника с кодом='+SE_Code.Text
                                                +' не найден !');
    end;
  end
  else begin
     ShowMessage('Справочник не открыт');
  end;
end;
```

На рис. 4.219 представлен вид формы при визуализации элемента справочника с кодом 1.

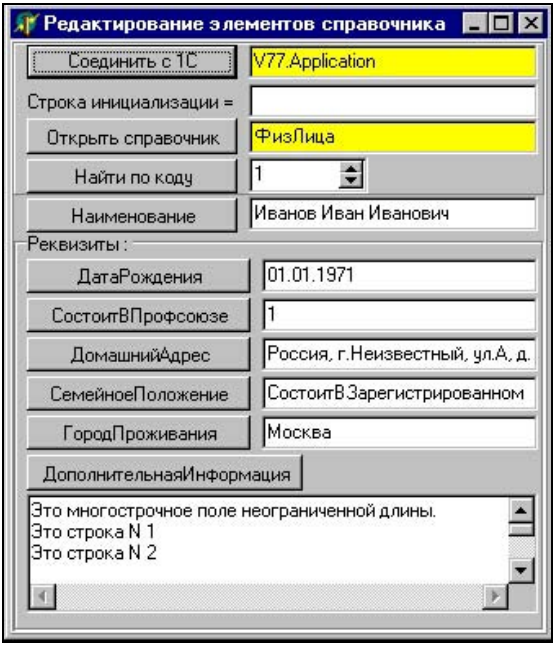



**Рис. 4.219.** Визуализация элемента справочника `ФизЛица`

Для сравнения на рис. 4.220 представлена визуализация этого же элемента в программе 1С:Предприятие.

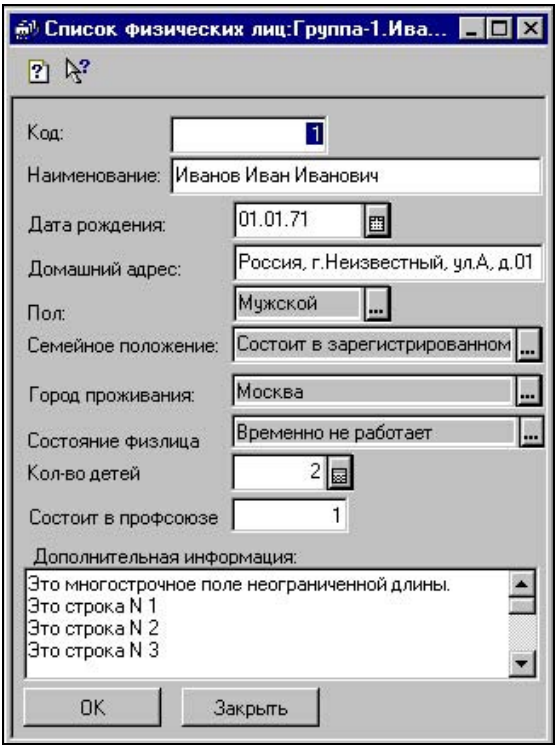



**Рис. 4.220.** Визуализация элемента справочника `ФизЛица` в программе 1С:Предприятие

В листинге 4.109 представлен текст обработчика события "нажатие кнопки **Наименование**".

**Листинг 4.109. Редактирование свойства `Наименование` элемента справочника**

```
procedure TForm1.SpeedButton7Click(Sender: TObject);
// Редактирование свойства "Наименование"
// элемента справочика "ФизЛица"
Var
  Sx: string;
begin
  // Проверяем, существует ли ссылка на справочник
  if not VarIsEmpty(ole_Ref) then begin
```

```
      Sx:=Edit_Name.Text;
      // Вводим новое значение
      if InputQuery((Sender as TSpeedButton).Caption,
                    'Новое значение',
                    Sx) then begin
         Sx:=Trim(Sx);
         // Редактируем значение в справочнике
         ole_Ref.Наименование:=Sx;
         // Записываем изменения в справочнике в базу данных
         ole_Ref.Записать;
         // Вновь читаем значение из справочника
         Edit_Name.Text:=Trim(ole_Ref.Наименование);
      end;
   end
   else begin
      ShowMessage('Справочник не открыт');
   end;
end;
```

Поясним текст обработчика.

В строке:

```
if InputQuery( ... ) then begin
```

вводится новое значение свойства `Наименование`.

В строке:

```
ole_Ref.Наименование:=Sx;
```

свойству `Наименование` элемента справочника присваивается новое значение.

В строке:

```
ole_Ref.Записать;
```

записываем изменения в элементе справочника в базу данных. Вот, собственно, и все.

В листинге 4.110 представлен текст обработчика события "нажатие кнопки **ДатаРождения**".

**Листинг 4.110. Редактирование реквизита `ДатаРождения` (тип `Дата`)**

```
procedure TForm1.SpeedButton2Click(Sender: TObject);
// Редактирование реквизита "ДатаРождения"
Var
```

```
  Sx: string;
begin
  // Проверяем, существует ли ссылка на справочник
  if not VarIsEmpty(ole_Ref) then begin
     Sx:=Edit_DR.Text;
     // Вводим новое значение
     if InputQuery((Sender as TSpeedButton).Caption,
                   'Новое значение',
                   Sx) then begin
        Sx:=Trim(Sx);
        // Редактируем значение реквизита
        ole_Ref.ДатаРождения:=Sx;
        // Записываем изменения в справочнике в базу данных
        ole_Ref.Записать;
        // Вновь читаем значение реквизита из справочника
        Edit_DR.Text:=Trim(ole_Ref.ДатаРождения);
     end;
  end
  else begin
     ShowMessage('Справочник не открыт');
  end;
end;
```

Реквизит `ДатаРождения` имеет тип `Дата`. Как видите, способ редактирования непериодических реквизитов такого типа аналогичен способу редактирования свойства `Наименование`. По такому принципу можно редактировать реквизиты всех базовых типов (`Число`, `Дата`, `Строка`), хотя для реквизитов типа `Число` корректнее использовать вариант:

```
ole_Ref.СостоитВПрофсоюзе:= StrToInt(Sx);
```

В случае, если нужно отредактировать реквизит типа `Строка неограниченной длины`, можно создать соответствующую форму (пример приведен на рис. 4.221).

При редактировании реквизита такого типа сначала значение реквизита присваивается свойству `Lines` Delphi-компонента `Memo1`:

```
Dlg_Edit_AttrString0.Memo1.Text:= ole_Ref.ДополнительнаяИнформация;
```

Затем вызывается метод `Execute` диалоговой формы и производится редактирование значения. После этого новое значение присваивается реквизиту:

```
ole_Ref.ДополнительнаяИнформация:= Dlg_Edit_AttrString0.Memo1.Text;
```

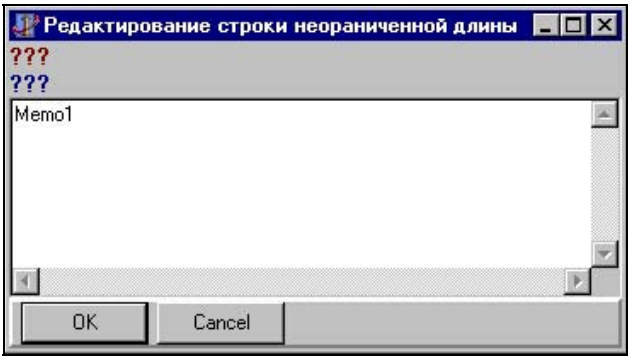



**Рис. 4.221.** Форма `Dlg_Edit_AttrString0` для редактирования реквизитов типа `Строка неограниченной длины`

Изменения сохраняются в базе данных:

```
ole_Ref.Записать;
```

То есть все аналогично предыдущему случаю.

Здесь следует отметить, что значения реквизитов агрегатного типа нельзя редактировать "напрямую" так, как мы это делали с реквизитами базового типа (используя функцию `InputQuery`). Поэтому нам нужен инструментарий для выбора значения агрегатного типа. Например, для реквизита типа `Перечисление` понадобится средство выбора конкретного значения заданного перечисления (среди штатных средств системы 1С:Предприятие для этого есть метод `ВвестиПеречисление`). Используя наработки, полученные в *разд. 4.4*, создадим (для "тренировочных" целей) простую диалоговую форму, которая позволит нам эмулировать штатный метод системы 1С:Предприятие `ВвестиПеречисление`.

На рис. 4.222 представлен внешний вид формы диалогового окна для выбора значения перечисления.

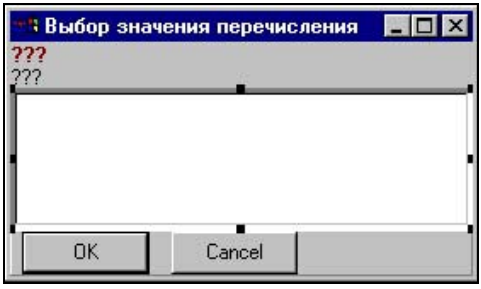



**Рис. 4.222.** Форма для выбора значения заданного перечисления

В листинге 4.111 представлен исходный текст метода `Execute` класса `TDlg_Select_ValueOfEnum`.

**Листинг 4.111. Метод `Execute` класса `TDlg_Select_ValueOfEnum` для выбора значения перечисления**

```
type
 TDlg_Select_ValueOfEnum = class(TForm)
   ...
   ...
 public
   { Public declarations }
   // Идентификатор выбранного значения перечисления
   idValueEnum: string;
   // Ссылка на выбранное значение перечисления как объект
   oleValueEnum: oleVariant;
   // Объявление метода Execute формы
   function Execute(Ole_1C_App: oleVariant; idEnum, idEnumVal: string):
boolean;
 end;
...
implementation
...
function TDlg_Select_ValueOfEnum.Execute(Ole_1C_App: oleVariant;
                                 idEnum, idEnumVal: string): boolean;
// Ole_1C_App — ссылка на 1С как сервер автоматизации OLE
// idEnum — идентификатор перечисления
// idEnumVal — идентификатор значения перечисления по умолчанию
Var
 OleEnum: oleVariant;
 i,CountVal: integer;
 idVal: string;
begin
 Result:=FALSE;
 ListBox1.Items.Clear;
 Label_Enum.Caption:='???';
 Label_EnumVal.Caption:='???';
 idValueEnum:='';
 oleValueEnum:= UnAssigned;
```

```
 // Проверяем, есть ли ссылка на 1С
 if not VarIsEmpty(Ole_1C_App) then begin
   idEnumVal:=Trim(idEnumVal);
   idEnum:=Trim(idEnum);
   if length(idEnum)>0 then begin
     Label_Enum.Caption:=idEnum;
     Label_EnumVal.Caption:=idEnumVal;
     // Получаем ссылку на объект "Перечисление" (идентификатор=idEnum)
     OleEnum:=Ole_1C_App.Перечисление.ПолучитьАтрибут(''+idEnum);
     // Получаем количество значений перечисления
     CountVal:= OleEnum.КоличествоЗначений*1;
     for i:=1 to CountVal do
      begin
       // Получаем идентификатор очередного значения перечисления
       idVal:=Trim(OleEnum.ЗначениеПоНомеру(i*1).Идентификатор);
       // Помещаем идентификатор в список
       ListBox1.Items.ADD(idVal);
       Application.ProcessMessages;
     end;
     // Позиционируем курсор в списке на значении idEnumVal
     ListBox1.ItemIndex:=ListBox1.Items.IndexOf(idEnumVal);
     if ShowModal = mrOk then begin
       if ListBox1.ItemIndex>=0 then begin
         // Идентификатор выбранного значения перечисления
         idValueEnum:=Trim(ListBox1.Items[ListBox1.ItemIndex]);
         // Ссылка на выбранное значение перечисления как объект
         oleValueEnum:= OleEnum.ЗначениеПоИдентификатору(''+idValueEnum);
         Result:=TRUE;
       end;
     end;
   end;
 end;
end;
end.
```

Здесь особых пояснений не требуется, поскольку эти вопросы (связанные с перечислениями) мы подробно рассматривали в *разд. 4.4*.

В листинге 4.112 представлен текст, иллюстрирующий способ редактирования реквизитов типа `Перечисление`.

**Листинг 4.112. Редактирование реквизита СемейноеПоложение**

```
procedure TForm1.SpeedButton5Click(Sender: TObject);
// Редактирование реквизита "СемейноеПоложение"
begin
  // Проверяем, существует ли ссылка на справочник
  if not VarIsEmpty(ole_Ref) then begin
     // Вызываем диалог для выбора значения перечисления
     if Dlg_Select_ValueOfEnum.Execute(ole_1C,
                          'СемейноеПоложение',
                          Edit_Fam.Text) then begin
        // Редактируем значение реквизита
        ole_Ref.СемейноеПоложение:=Dlg_Select_ValueOfEnum.oleValueEnum;
        // Записываем изменения в справочнике в базу данных
        ole_Ref.Записать;
        // Вновь читаем значение реквизита из справочника
        Edit_Fam.Text:=Trim(ole_Ref.СемейноеПоложение.Идентификатор);
     end;
  end
  else begin
     ShowMessage('Справочник не открыт');
  end;
end;
```

Как видите, подход здесь такой же, как и при редактировании реквизитов базовых типов. Единственное отличие — вместо функции `InputQuery` мы используем разработанную нами диалоговую форму для выбора значения перечисления (рис. 4.222). Здесь следует отметить важную деталь. В данном случае мы присваиваем реквизиту (агрегатного типа) справочника значение, хранящееся в переменной типа `oleVariant` (или `Variant`), не заботясь о том, каким образом происходит его "трансформация" в соответствующее значение объектного типа (в контексте системы 1С:Предприятие):

```
ole_Ref.СемейноеПоложение:=Dlg_Select_ValueOfEnum.oleValueEnum;
```

Точно так же редактируются реквизиты типа `Справочник`. Только для выбора значения реквизита (из множества элементов соответствующего справочника) нужно использовать метод `Выбрать` того справочника, откуда выбираются значения для реквизита (или написать собственное средство для выбора соответствующего значения). Здесь мы на этом останавливаться не будем,

но в рамках данного примера разработана форма (для "тренировочных" целей) выбора значений для реквизитов типа `Справочник`.

В листинге 4.113 представлен текст, иллюстрирующий способ редактирования реквизитов типа `Справочник`.

**Листинг 4.113. Редактирование реквизита `ГородПроживания`**

```
procedure TForm1.SpeedButton6Click(Sender: TObject);
// Редактирование реквизита "ГородПроживания"
begin
  // Проверяем, существует ли ссылка на справочник
  if not VarIsEmpty(ole_Ref) then begin
     // Вызываем диалог для выбора элемента справочника
     if Dlg_Select_ElemFromRef.Execute(ole_1C,
                    'ГородаСтранМира',
                    ole_Ref.ГородПроживания.Код) then begin
        // Редактируем значение реквизита
        ole_Ref.ГородПроживания:=Dlg_Select_ElemFromRef.oleElemSel;
        // Записываем изменения в справочнике в базу данных
        ole_Ref.Записать;
        // Вновь читаем значение реквизита из справочника
        Edit_Gorod.Text:=Trim(ole_Ref.ГородПроживания.Наименование);
     end;
  end
  else begin
     ShowMessage('Справочник не открыт');
  end;
end;
```

Здесь для выбора нового значения реквизита типа `Справочник` вызывается метод `Execute` класса `TDlg_Select_ElemFromRef`. В остальном — все то же самое (новое значение присваивается реквизиту):

```
ole_Ref.ГородПроживания:=Dlg_Select_ElemFromRef.oleElemSel;
```

Скомпилируем наш Delphi-проект, запустим его на выполнение (см. рис. 4.219) и попробуем изменить значения реквизитов элемента справочника с кодом 1.

На рис. 4.223 представлен процесс редактирования свойства `Наименование` элемента справочника (см. листинг 4.109).

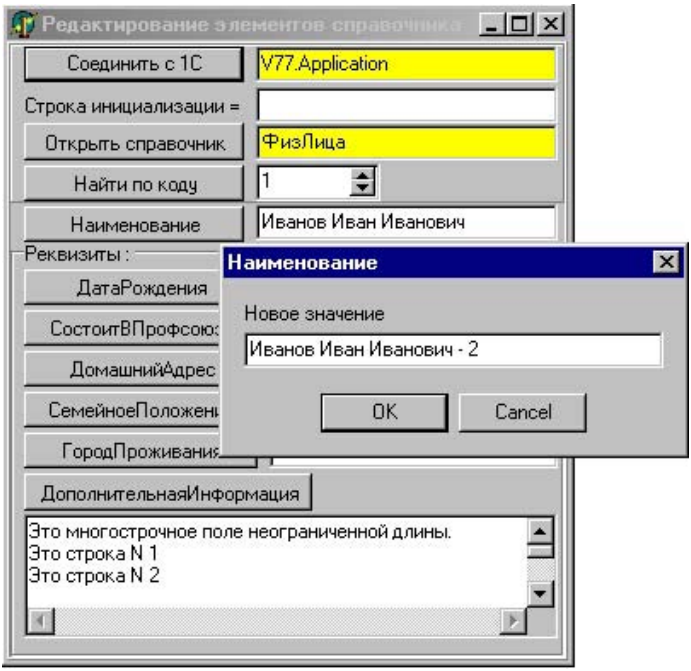



**Рис. 4.223.** Редактирование свойства `Наименование` элемента справочника `ФизЛица`

На рис. 4.224 представлен процесс редактирования реквизита `ДатаРождения` элемента справочника (см. листинг 4.110).

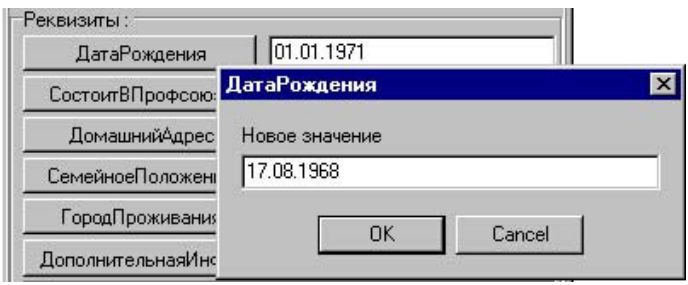



**Рис. 4.224.** Редактирование реквизита `ДатаРождения` элемента справочника `ФизЛица`

На рис. 4.155 представлен процесс редактирования реквизита `СемейноеПоложение` элемента справочника (см. листинг 4.112).

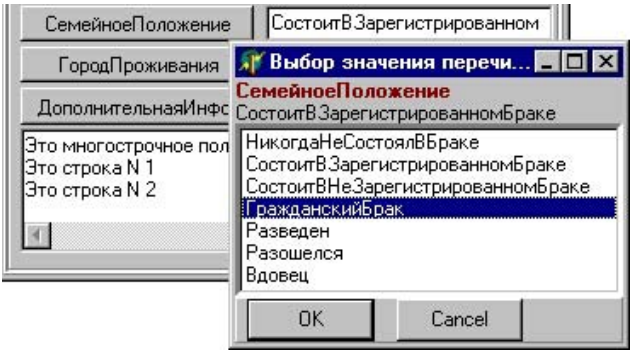



**Рис. 4.225.** Редактирование реквизита `СемейноеПоложение` элемента справочника `ФизЛица`

На рис. 4.226 представлен процесс редактирования реквизита `ГородПроживания` элемента справочника (см. листинг 4.113).

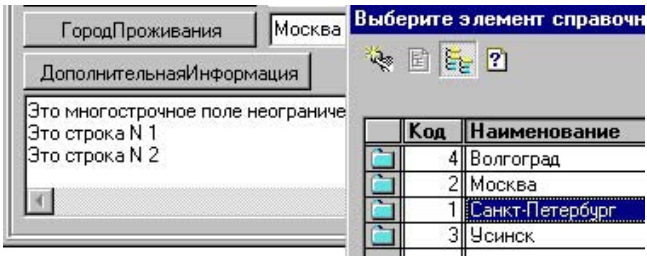



**Рис. 4.226.** Редактирование реквизита `ГородПроживания` элемента справочника `ФизЛица`

На рис. 4.227 представлен процесс редактирования реквизита `ДополнительнаяИнформация` элемента справочника.

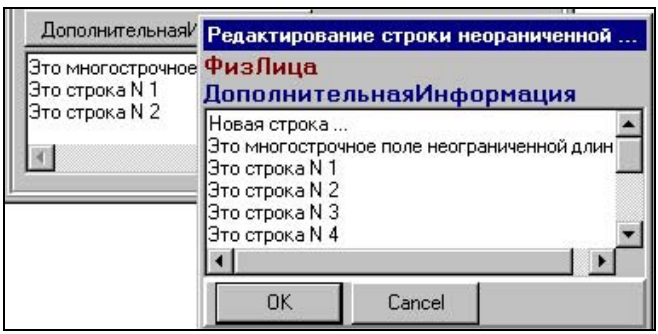



**Рис. 4.227.** Редактирование реквизита `ДополнительнаяИнформация` элемента справочника `ФизЛица`

На рис. 4.228 представлен вид формы при визуализации элемента справочника с кодом 1 после редактирования.

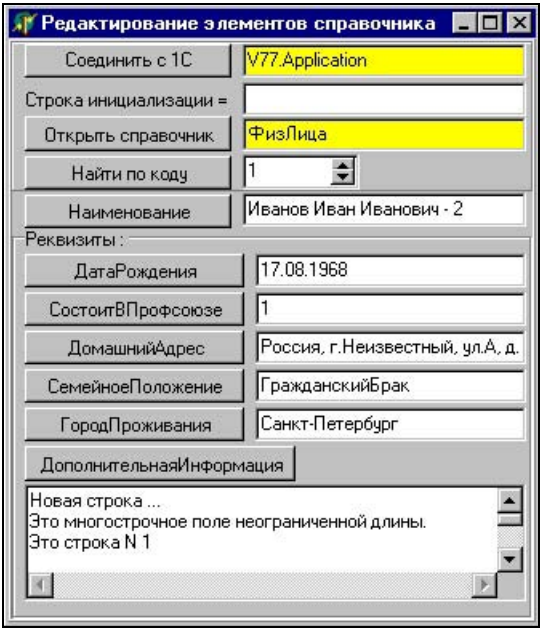



**Рис. 4.228.** Визуализация элемента справочника `ФизЛица` после редактирования

При конфигурировании информационной структуры 1С:Предприятие для реквизитов справочников может быть указан тип `Неопределенный`. Это значит, что такому реквизиту можно программно назначить как базовый тип (`Число`, `Строка`, `Дата`), так и агрегатный (`Справочник`, `Перечисление`, `Документ` и т. д.). По метаданным такой реквизит будет *всегда* "проходить" как реквизит неопределенного типа (или вида), но тип (вид) его *значения* для каждого элемента справочника может отличаться.

Откроем Конфигуратор 1С:Предприятие и в "нашей" конфигурации создадим новый справочник (рис. 4.229).

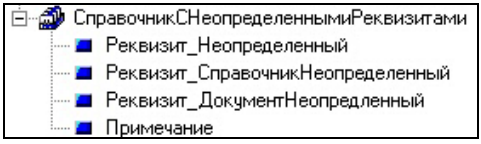



**Рис. 4.229.** Новый справочник `СправочникСНеопределеннымиРеквизитами`. Реквизиты неопределенного типа

***Таблица 4.13.*** *Реквизиты справочника* `СправочникСНеопределенныМиРеквизитами`

| Реквизит | Тип |
|---|---|
| `Реквизит_Неопределенный` | `Неопределенный` |
| `Реквизит_СправочникНеопределенный` | `Справочник` неопределенного вида |
| `Реквизит_ДокументНеопредленный` | `Документ` неопределенного вида |
| `Примечание` | `Строка` (64 символа) |

Сохраним изменения в конфигурации, запустим программу 1С:Предприятие и создадим 6 элементов в справочнике `СправочникСНеопределенныМиРеквизитами` (рис. 4.230).

| | Код | Наимен... | Реквизит_Не... | Реквизит_Спр... | Реквизит_Док... | Приме |
|---|---|---|---|---|---|---|
| | 1 | Элемент-1 | | | | |
| | 2 | Элемент-2 | | | | |
| | 3 | Элемент-3 | | | | |
| | 4 | Элемент-4 | | | | |
| | 5 | Элемент-5 | | | | |
| | 6 | Элемент-6 | | | | |

**Рис. 4.230.** Элементы справочника `СправочникСНеопределенныМиРеквизитами`

Попробуем, ради интереса, ввести какие-то значения в столбец реквизита `Реквизит_Неопределенный`. Думаю, что это вряд ли получится (до тех пор, пока тип реквизита — `Неопределенный`)…

Создадим новый Delphi-проект (sp_EditAttrInRef_2.DPR).

***Примечание***

Исходные тексты находятся на сопроводительном компакт-диске книги в папке \Part_1\Chapter_04\Topic_4_7\Item_4_7_1\sp_EditAttrInRef_2.PAS\01.

По аналогии с проектом sp_EditAttrInRef.DPR попробуем визуализировать справочник `СправочникСНеопределенныМиРеквизитами` (листинг 4.114).

**Листинг 4.114. Визуализация справочника `СправочникСНеопределенныМиРеквизитами`**

```
procedure TForm1.SpeedButton9Click(Sender: TObject);
// Обработчик события "нажатие кнопки "НайтиПоКоду"
// Визуализация элемента справочника
```

```
begin
  ...
  //-------------------------------------
  // Читаем информацию из справочника
  Edit_Name.Text:=Trim(ole_Ref.Наименование);
  Edit_N.Text:=Trim(ole_Ref.Реквизит_Неопределенный);
Edit_NR.Text:=
      Trim(ole_Ref.Реквизит_СправочникНеопределенный.Наименование);
  Edit_ND.Text:=Trim(ole_Ref.Реквизит_ДокументНеопределенный.Вид)
              + '  N '
              + Trim(ole_Ref.Реквизит_ДокументНеопределенный.НомерДок)
              + ' ['
              + Trim(ole_Ref.Реквизит_ДокументНеопределенный.ДатаДок)
              + ']';
  Edit_Prim.Text:=Trim(ole_Ref.Примечание);
  //-------------------------------------
  ...
end;
```

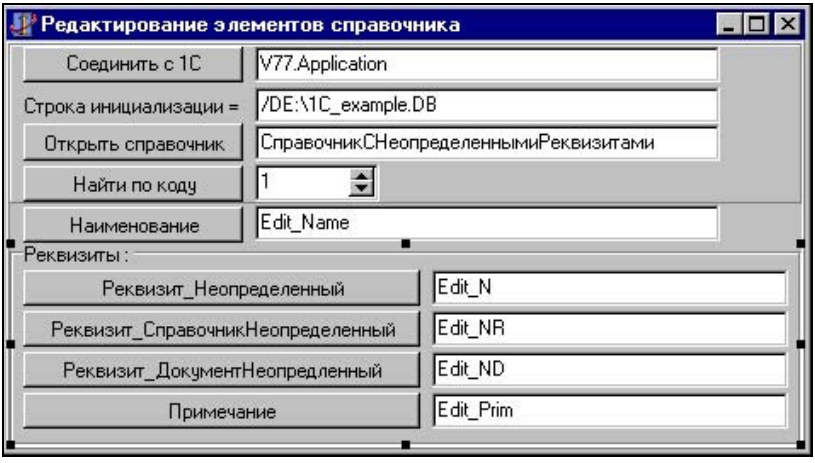



**Рис. 4.231.** Форма проекта sp_EditAttrInRef_2.DPR

Результат будет плачевным (рис. 4.232).

Причина очевидна. И простого решения здесь, вероятно, нет.

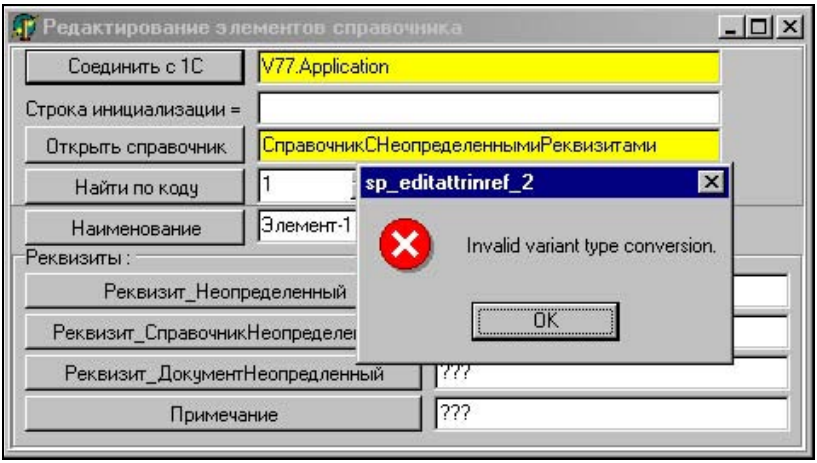



**Рис. 4.232.** Попытка прочитать значения реквизитов неопределенного типа

Один из возможных вариантов решения этой проблемы (далеко не идеальный) — создать в глобальном модуле конфигурации функцию, считывающую значение реквизита справочника и конвертирующую его в строковое значение, а затем вызывать эту функцию (в контексте метода `EvalExpr`) для всех реквизитов (или только для тех, которые по метаданным "проходят" как реквизиты неопределенного типа).

Возможный вариант такой функции представлен в листинге 4.115.

**Листинг 4.115. Функция в глобальном модуле 1С, считывающая значение реквизита справочника**

```
// Вызов функции гл9_СокрЛПдляЗначРеквизитаСправочника():
// идСправочника — идентификатор справочника,
// идРеквизита — идентифкатор реквизита,
// КодЭлемента — код элемента справочника,
// ФлагПоиска:
//   0 — во всем справочнике,
//   1 — внутри родителя,
//   2 — по полному коду (с учетом всех родителей).
Функция гл9_СокрЛПдляЗначРеквизитаСправочника(идСправочника,
                                              идРеквизита,
                                              КодЭлемента,
                                              ДатаДляПериод="",
                                              ФлагПоиска=0) Экспорт
```

```
  Res = "";
  идСправочника = СокрЛП(идСправочника);
  идРеквизита = СокрЛП(идРеквизита);
  Если (СтрДлина(идСправочника)>0) И (СтрДлина(идРеквизита)>0) Тогда
    // Проверяем существовние справочника <идСправочника>
    Если Метаданные.Справочник(""+идСправочника).Выбран()>0 Тогда
      // Проверяем существовние реквизита <идРеквизита>
      // справочника <идСправочника>
МетаРеквизит =
  Метаданные.Справочник(""+идСправочника).Реквизит(""+идРеквизита);
      Если МетаРеквизит.Выбран()>0 Тогда
        Спр = СоздатьОбъект("Справочник."+идСправочника);
        Если Спр.НайтиПоКоду(КодЭлемента,ФлагПоиска)>0 Тогда
          Если МетаРеквизит.Периодический>0 Тогда
            // Если реквизит периодический — нужно установить дату,
            // на которую следует выбрать значение
            sDate = ДАТА(ДатаДляПериод);
            // Проверяем параметр (дату) на корректность
            Если ДатаГод(sDate)<=0 Тогда
              // Дата не корректна — установим дату на "текущую дату"
              sDate = ТекущаяДата();
            КонецЕсли;
            // Установим дату выборки значения период. реквизита
            Спр.ИспользоватьДату(sDate,1);
          КонецЕсли;
          // Читаем значение реквизита из справочника
          ЗначРекв=Спр.ПолучитьАтрибут(""+идРеквизита);
          // Применяем СокрЛП()
          Res = СокрЛП(ЗначРекв);
          Если стрДлина(Res)<=0 Тогда
            // Если значение — пустая строка, формируем
            // строковое представление по типу значения
            ТипЗн=ТипЗначения(ЗначРекв);
            Если ТипЗн=0 Тогда
               Res="<Неопределенный>";
            КонецЕсли;
            Если ТипЗн=10 Тогда
               Res="<Перечисление>";
```

```
            КонецЕсли;
            Если ТипЗн=11 Тогда
               Res="<Справочник>";
            КонецЕсли;
            Если ТипЗн=12 Тогда
               Res="<Документ>";
            КонецЕсли;
          КонецЕсли;
        КонецЕсли;
      КонецЕсли;
    КонецЕсли;
  КонецЕсли;
  Возврат Res;
КонецФункции // гл9_СокрЛПдляЗначРеквизитаСправочника
```

В листинге 4.116 представлен пример вызова функции глобального модуля конфигурации (`гл9_СокрЛПдляЗначРеквизитаСправочника`) из Delphi для нашего примера.

**Листинг 4.116. Визуализация справочника с использованием функции `гл9_СокрЛПдляЗначРеквизитаСправочника`**

```
procedure TForm1.SpeedButton9Click(Sender: TObject);
// Обработчик события "нажатие кнопки "НайтиПоКоду"
// Визуализация элемента справочника
Var
  sEvalExpr: string; // Параметр для <EvalExpr>
begin
  ...
  Edit_Name.Text:=Trim(ole_Ref.Наименование);
  sEvalExpr:='гл9_СокрЛПдляЗначРеквизитаСправочника(';
  sEvalExpr:=sEvalExpr+'"'+Edit_RefName.Text+'",';
  sEvalExpr:=sEvalExpr+'"'+'Реквизит_Неопределенный'+'",';
  sEvalExpr:=sEvalExpr+SE_Code.Text+',';
  sEvalExpr:=sEvalExpr+'"",';
  sEvalExpr:=sEvalExpr+'0';
  sEvalExpr:=sEvalExpr+')';
  Edit_N.Text:=Trim(ole_1C.EvalExpr(sEvalExpr));
  sEvalExpr:='гл9_СокрЛПдляЗначРеквизитаСправочника(';
```

```
  sEvalExpr:=sEvalExpr+'"'+Edit_RefName.Text+'",';
  sEvalExpr:=sEvalExpr+'"'+'Реквизит_СправочникНеопределенный'+'",';
  sEvalExpr:=sEvalExpr+SE_Code.Text+',';
  sEvalExpr:=sEvalExpr+'"",';
  sEvalExpr:=sEvalExpr+'0';
  sEvalExpr:=sEvalExpr+')';
  Edit_NR.Text:=Trim(ole_1C.EvalExpr(sEvalExpr));
  sEvalExpr:='гл9_СокрЛПдляЗначРеквизитаСправочника(';
  sEvalExpr:=sEvalExpr+'"'+Edit_RefName.Text+'",';
  sEvalExpr:=sEvalExpr+'"'+'Реквизит_ДокументНеопределенный'+'",';
  sEvalExpr:=sEvalExpr+SE_Code.Text+',';
  sEvalExpr:=sEvalExpr+'"",';
  sEvalExpr:=sEvalExpr+'0';
  sEvalExpr:=sEvalExpr+')';
  Edit_ND.Text:=Trim(ole_1C.EvalExpr(sEvalExpr));
  sEvalExpr:='гл9_СокрЛПдляЗначРеквизитаСправочника(';
  sEvalExpr:=sEvalExpr+'"'+Edit_RefName.Text+'",';
  sEvalExpr:=sEvalExpr+'"'+'Примечание'+'",';
  sEvalExpr:=sEvalExpr+SE_Code.Text+',';
  sEvalExpr:=sEvalExpr+'"",';
  sEvalExpr:=sEvalExpr+'0';
  sEvalExpr:=sEvalExpr+')';
  Edit_Prim.Text:=Trim(ole_1C.EvalExpr(sEvalExpr));
  ...
end;
```

На рис. 4.233 представлен результат визуализации справочника `СправочникСНеопределенными Реквизитами` с помощью функции `гл9_СокрЛПдляЗначРеквизитаСправочника`.

Среди штатных средств языка программирования 1С:Предприятие есть функции `ТипЗначенияСтр` и `ТипЗначения`. Этим функциям в качестве параметра передается какое-то значение (реквизита или переменной). После вызова обе эти функции возвращают тип значения. Можно было бы этим воспользоваться, чтобы при визуализации "обходить" реквизиты, *значения* которых имеют неопределенный тип. Но вызвать эти функции посредством механизма OLE Automation, написав что-то вроде:

```
ole_1C.ТипЗначенияСтр(...);
ole_1C.ТипЗначения(...);
```

не получится.

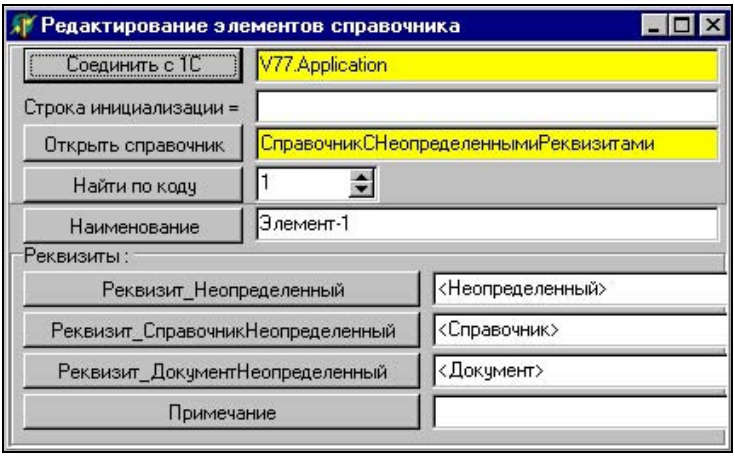



**Рис. 4.233.** Визуализация справочника с помощью функции `гл9_СокрЛПдляЗначРеквизитаСправочника`

Можно для их вызова воспользоваться методом `EvalExpr`, но, опять же, явно передать в качестве параметра значение агрегатного типа (`oleVariant`) этим функциям нельзя.

Одно из возможных решений (тоже далеко не идеальное) — создать в глобальном модуле конфигурации соответствующую функцию, которую можно было бы вызывать (через OLE Automation) с помощью метода `EvalExpr`.

Возможный вариант такой функции приведен в листинге 4.117.

**Листинг 4.117. Функция в глобальном модуле 1С, определяющая тип значения реквизита справочника**

```
// Вызов функции гл9_ТипЗначения_ДляЗначРеквизитаСправочника():
// идСправочника — идентификатор справочника,
// идРеквизита — идентифкатор реквизита,
// КодЭлемента — код элемента справочника,
// ФлагПоиска:
//   0 — во всем справочнике,
//   1 — внутри родителя,
//   2 — по полному коду (с учетом всех родителей).
// Возвращаемое значение:
//  -1 — некорректые входные параметры
//  >=0 — код типа данных (как его возвращает функция "ТипЗначения").
```

```
Функция гл9_ТипЗначения_ДляЗначРеквизитаСправочника(
                                        идСправочника,
                                        идРеквизита,
                                        КодЭлемента,
                                        ДатаДляПериод="",
                                        ФлагПоиска=0) Экспорт
 Res = -1;
 идСправочника = СокрЛП(идСправочника);
 идРеквизита = СокрЛП(идРеквизита);
 Если (СтрДлина(идСправочника)>0) И (СтрДлина(идРеквизита)>0) Тогда
   // Проверяем существование справочника <идСправочника>
   Если Метаданные.Справочник(""+идСправочника).Выбран()>0 Тогда
     // Проверяем существование реквизита <идРеквизита> справочника
МетаРеквизит=
  Метаданные.Справочник(""+идСправочника).Реквизит(""+идРеквизита);
     Если МетаРеквизит.Выбран()>0 Тогда
       Спр = СоздатьОбъект("Справочник."+идСправочника);
       // Находим элемент справочника с соответствующим кодом
       Если Спр.НайтиПоКоду(КодЭлемента,ФлагПоиска)>0 Тогда
         Если МетаРеквизит.Периодический>0 Тогда
           // Если реквизит периодический — нужно установить дату,
           // на  которую следует выбрать значение
           sDate = ДАТА(ДатаДляПериод);
           // Проверяем параметр (дату) на корректность
           Если ДатаГод(sDate)<=0 Тогда
             // Дата не корректна — будем ориентироваться
             // на "текущую дату"
             sDate = ТекущаяДата();
           КонецЕсли;
           // Установим дату выборки значения период. реквизита
           Спр.ИспользоватьДату(sDate,1);
         КонецЕсли;
         // Читаем значение реквизита из справочника
         V = Спр.ПолучитьАтрибут(""+идРеквизита);
         // Определяем его тип
         Res = ТипЗначения(V);
       КонецЕсли;
     КонецЕсли;
```

```
   КонецЕсли;
 КонецЕсли;
 Возврат Res;
КонецФункции // гл9_ТипЗначения_ДляЗначРеквизитаСправочника
```

Вызывая эту функцию с помощью метода `EvalExpr`:

```
sEvalExpr:=' гл9_ТипЗначения_ДляЗначРеквизитаСправочника(';
sEvalExpr:=sEvalExpr+'"'+Edit_RefName.Text+'",';
sEvalExpr:=sEvalExpr+'"'+'Реквизит_Неопределенный'+'",';
sEvalExpr:=sEvalExpr+SE_Code.Text+',';
sEvalExpr:=sEvalExpr+'"",';
sEvalExpr:=sEvalExpr+'0';
sEvalExpr:=sEvalExpr+')';
NrType:= ole_1C.EvalExpr(sEvalExpr)*1;
if NrType>0 then begin
   ...
end;
```

можно игнорировать реквизиты, значения которых имеют тип `Неопределенный`.

Еще один вариант частичного решения проблемы — использование Delphi-функции `VarType`. Функция `VarType` (модуль system.PAS) в качестве параметра принимает значение типа `oleVariant` (или `Variant`) и возвращает результат типа `Integer` (множество значений которого определено в виде констант в модуле system.PAS).

Перепишем фрагмент текста (см. листинг 4.114), как показано в листинге 4.118.

**Листинг 4.118. Визуализация справочника `СправочникСНеопределеннымиРеквизитами` (другой вариант)**

```
procedure TForm1.SpeedButton9Click(Sender: TObject);
// Обработчик события "нажатие кнопки "НайтиПоКоду"
// Визуализация элемента справочника
begin
  ...
  //-------------------------------------
  // Читаем тип данных, которые возвращает функция "VarType"
  Edit_Name.Text:=IntToStr(VarType(ole_Ref.Наименование));
  Edit_N.Text:=IntToStr(VarType(ole_Ref.Реквизит_Неопределенный));
```

```
  Edit_NR.Text:=
     IntToStr(VarType(ole_Ref.Реквизит_СправочникНеопределенный));
  Edit_ND.Text:=
     IntToStr(VarType(ole_Ref.Реквизит_ДокументНеопределенный));
  Edit_Prim.Text:=IntToStr(VarType(ole_Ref.Примечание));
  //-------------------------------------
  ...
end;
```

Результат представлен на рис. 4.234.

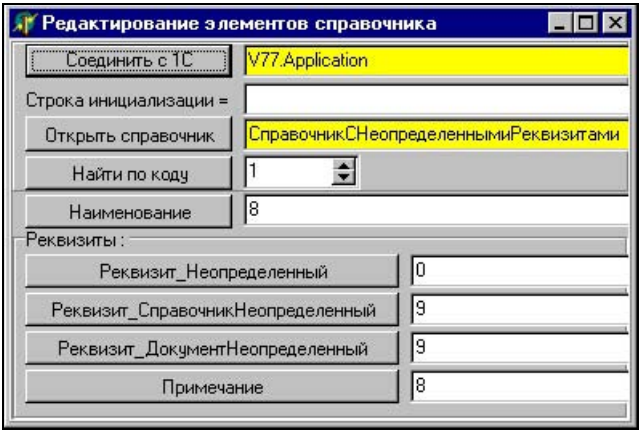



**Рис. 4.234.** Результат использования функции `VarType` для справочника `СправочникСНеопределеннымиРеквизитами`

Проведем тот же эксперимент со справочником `ФизЛица` (проект sp_EditAttrInRef.DPR). Результат представлен на рис. 4.235.

В табл. 4.14 приведены константы, соответствующие значениям, которые функция `VarType` вернула в нашем случае.

***Таблица 4.14.*** *Результат, который вернула функция* `VarType`

| **Значение** | **Константа** (в модуле system.PAS) | **Тип реквизита справочника** |
|---|---|---|
| $0000 | varEmpty | Неопределенный |
| $0005 | varDouble | Число |
| $0007 | varDate | Дата |
| $0008 | varOleStr | Строка |
| $0009 | varDispatch | Агрегатный |

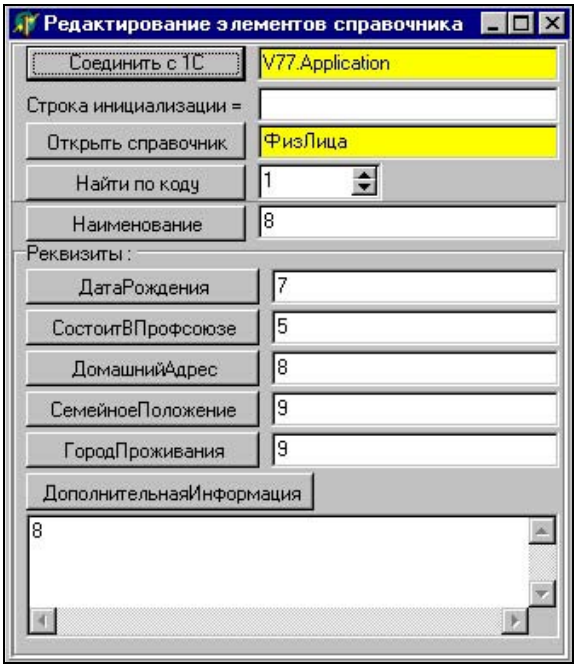



**Рис. 4.235.** Результат использования функции `VarType` для справочника `ФизЛица`

То есть с помощью функции `VarType` можно при визуализации исключить обращение к реквизитам, *значения* которых имеют тип `Неопределенный`. Более того, для значений реквизитов, имеющих базовый тип, функция `VarType` возвращает однозначный результат, что можно использовать при визуализации справочников.

Для того чтобы можно было редактировать реквизит неопределенного типа, нужно сначала назначить ему какой-то тип (базовый или агрегатный). Ранее было сказано, что реквизиту неопределенного типа можно назначить какой-либо тип с помощью метода `НазначитьТип` (или `SetType`) справочника. Его синтаксис:

`НазначитьТип(`*`<ИмяРеквизита>`*`, `*`<ИмяТипа>`*`, `*`<Длина>`*`, `*`<Точность>`*`)`

Параметры: *`<ИмяРеквизита>`* (строковое выражение) — идентификатор реквизита (неопределенного типа) справочника, как он назван в Конфигураторе; *`<ИмяТипа>`* (строковое выражение) — тип данных (или Вид субконто), который назначается реквизиту справочника (например: `Строка`, `Число`, `Справочник.Товары`, `Документ.РасходнаяНакладная` и т. п.); *`<Длина>`* (числовое выражение, необязательный) — длина поля представления данных (имеет смысл только при задании числового или строкового типа); *`<Точность>`* (числовое выражение; необязательный) — количество знаков

после десятичной точки (имеет смысл только при задании числового типа). Метод НазначитьТип позволяет назначить тип для реквизита, которому в Конфигураторе назначен тип Неопределенный.

> ***Внимание!***
>
> Естественно, что перед вызовом метода НазначитьТип ссылка на справочник должна быть корректной и справочник должен быть спозиционирован на элементе, к реквизитам которого применяется метод НазначитьТип.

В листинге 4.119 приведены примеры вызова метода НазначитьТип справочника для реквизитов неопределенного типа.

**Листинг 4.119. Использование метода НазначитьТип справочника**

```
...
ole_Ref.НазначитьТип(''+idAttr, ''+'Число', 10, 2);
ole_Ref.НазначитьТип(''+idAttr, ''+'Строка', 25, 0);
ole_Ref.НазначитьТип(''+idAttr, ''+'Дата', 0, 0);
ole_Ref.НазначитьТип(''+idAttr, ''+'Справочник.ФизЛица', 0, 0);
ole_Ref.НазначитьТип(''+idAttr, ''+'Документ.ВыплатыФизЛицам', 0, 0);
ole_Ref.НазначитьТип(''+idAttr,
                     ''+'Перечисление.СпособВыплатыФизЛицам', 0, 0);
ole_Ref.НазначитьТип(''+idAttr, ''+'Неопределенный', 0, 0);
...
// Изменения в элементе справочника должны быть
// записаны в базу данных
ole_1C_ref.Записать;
```

Поясним листинг 4.119. Здесь:

- ole_Ref — ссылка на какой-то справочник (переменная типа oleVariant);
- idAttr — идентификатор реквизита неопределенного типа.

В общем-то, как видно, ничего сложного здесь нет в том случае, если мы точно знаем, что для данного справочника (ole_Ref) реквизит id_Attr имеет тип Неопределенный.

> ***Примечание***
>
> Для значений реквизитов неопределенного типа в 1С отведено только 22 символа. Поэтому команда (листинг 4.11):
>
> ```
> ole_Ref.НазначитьТип(''+idAttr, ''+ 'Строка', 25, 0);
> ```
>
> выполнится, но реально под значение реквизита будет выделено только *22* символа. Более подробно об этом "феномене" — в *части II* этой книги.

Модифицируем наш Delphi-проект (sp_EditAttrInRef_2.DPR).

**Примечание**

Исходные тексты находятся на сопроводительном компакт-диске книги в папке \Part_1\Chapter_04\Topic_4_7\Item_4_7_1\sp_EditAttrInRef_2.PAS\02.

Добавим в форму проекта кнопку **Тип** (рис. 4.236).

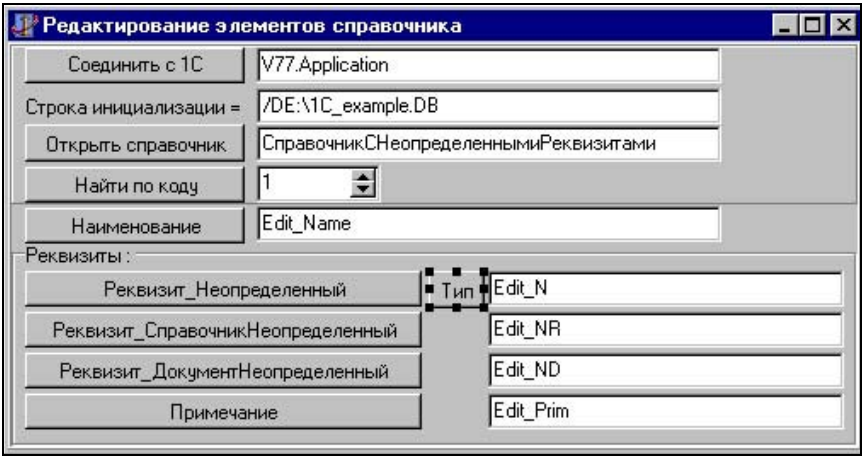



**Рис. 4.236.** Форма проекта sp_EditAttrInRef_2.DPR

Ее назначение — вызов метода `НазначитьТип` справочника для реквизита `Реквизит_Неопределенный`. Текст обработчика события "нажатие кнопки **Тип**" представлен в листинге 4.120.

**Листинг 4.120. Обработчик события "нажатие кнопки *Тип*"**

```
procedure TForm1.SpeedButton6Click(Sender: TObject);
// Вызов метода "НазначитьТип"
Var
  sType: string;       // Тип значения реквизита
  sEvalExpr: string;   // Параметр для EvalExpr
  CountChar,           // Количество символов
  Precis   : integer;  // Точность (количество знаков после запятой)
  Yes: byte;           // Вспомогательная переменная
  Sx: string;          // Вспомогательная переменная
begin
 // Проверяем, существует ли ссылка на справочник
```

```
if not VarIsEmpty(ole_Ref) then begin
  Yes:=0;
  CountChar:=0;
  Precis:=0;
  //-----------------------------
  // Определяем текущий тип значения реквизита
  sEvalExpr:='гл9_ТипЗначенияСтр_ДляЗначРеквизитаСправочника(';
  sEvalExpr:=sEvalExpr+'"'+Edit_RefName.Text+'",';
  sEvalExpr:=sEvalExpr+'"'+'Реквизит_Неопределенный'+'",';
  sEvalExpr:=sEvalExpr+SE_Code.Text+',';
  sEvalExpr:=sEvalExpr+'"",';
  sEvalExpr:=sEvalExpr+'0,';
  sEvalExpr:=sEvalExpr+'1';
  sEvalExpr:=sEvalExpr+')';
  // ShowMessage(sEvalExpr);
  sType:=Trim(ole_1C.EvalExpr(sEvalExpr));
  // ShowMessage(sType);
  //-----------------------------
  //-----------------------------
  // Указываем новый тип реквизита
  if InputQuery('НазначитьТип',
          'Новый тип',sType) then begin
    sType:=Trim(sType);
    if length(sType)>0 then begin
      Yes:=1;
      if UpperCase(sType)=UpperCase('Число') then begin
        // Новый тип реквизита - "Число"
        Yes:=0;
        Sx:='10';
        if InputQuery('Тип <Число>',
                 'Кол-во цифр всего',Sx) then begin
          Sx:=Trim(Sx);
          CountChar:=StrToInt(Sx);
          if CountChar>0 then begin
            Sx:='2';
            if InputQuery('Тип <Число>',
                 'Кол-во цифр после запятой',Sx) then begin
              Sx:=Trim(Sx);
              Precis:=StrToInt(Sx);
```

```
              Yes:=1;
            end;
          end;
        end;
      end;
      if UpperCase(sType)=UpperCase('Строка') then begin
        // Новый тип реквизита - "Строка"
        Yes:=0;
        Sx:='25';
        if InputQuery('Тип <Строка>',
               'Кол-во символов всего',Sx) then begin
          Sx:=Trim(Sx);
          CountChar:=StrToInt(Sx);
          if CountChar>0 then begin
            Yes:=1;
          end;
        end;
      end;
    end;
  end;
  //-----------------------------
  if Yes>0 then begin
    // Назначаем новый тип реквизита
    ole_Ref.НазначитьТип(''+'Реквизит_Неопределенный',
                         ''+sType,
                         CountChar*1,
                         Precis*1);
    // Записываем изменения в базу данных
    ole_Ref.Записать;
    // Вновь визуализируем элемент справочника
    SpeedButton9.Click;
  end;
 end;
end;
```

Поясним текст обработчика.

Сначала определяем текущий тип значения реквизита справочника. Для этого вызываем (с помощью метода `EvalExpr`) функцию глобального модуля `гл9_ТипЗначенияСтр_ДляЗначРеквизитаСправочника` (ее текст приведен далее).

Затем вводим новый тип реквизита (как показано в листинге 4.121). Если новый тип Число или Строка, то запрашиваем количество символов и точность (для типа Число). Далее вызываем метод справочника НазначитьТип и изменения в элементе справочника записываем на диск. Вот, собственно, и все.

Обратите внимание на вызов метода НазначитьТип:

```
ole_Ref.НазначитьТип(''+'Реквизит_Неопределенный',
                     ''+sType,
                     CountChar*1,
                     Precis*1);
```

Здесь (как и раньше) нужно в передаваемых параметрах указывать явно, что строковый параметр — это строка (параметр предваряется одинарными кавычками), а параметр типа Integer — это число (умножаем на 1).

**Листинг 4.121. Функция гл9_ТипЗначенияСтр_ДляЗначРеквизитаСправочника**

```
// Вызов функции гл9_ТипЗначенияСтр_ДляЗначРеквизитаСправочника():
// идСправочника — идентификатор справочника,
// идРеквизита — идентификатор реквизита,
// КодЭлемента — код элемента справочника,
// ФлагПоиска:
//   0 — во всем справочнике,
//   1 — внутри родителя,
//   2 — по полному коду (с учетом всех родителей).
// ВозвращатьВид  — если >0, то для агрегатных типов возвращается Вид
(через точку после типа).
// Примеры:
//   Дата
//   Перечисление.СостояниеФизлица
//   Справочник.ГородаСтранМира
Функция гл9_ТипЗначенияСтр_ДляЗначРеквизитаСправочника(
                                        идСправочника,
                                        идРеквизита,
                                        КодЭлемента,
                                        ДатаДляПериод="",
                                        ФлагПоиска=0,
                                        ВозвращатьВид=0) Экспорт
 Res = "";
 идСправочника = СокрЛП(идСправочника);
 идРеквизита = СокрЛП(идРеквизита);
```

```
 Если (СтрДлина(идСправочника)>0) И (СтрДлина(идРеквизита)>0) Тогда
   // Проверяем существовние справочника "идСправочника"
   Если Метаданные.Справочник(""+идСправочника).Выбран()>0 Тогда
     // Проверяем существовние реквизита "идРеквизита"
     // справочника "идСправочника"
МетаРеквизит = Метаданные.Справочник(""+идСправочника).Реквизит
  (""+идРеквизита);
     Если МетаРеквизит.Выбран()>0 Тогда
       Спр = СоздатьОбъект("Справочник."+идСправочника);
       Если Спр.НайтиПоКоду(КодЭлемента,ФлагПоиска)>0 Тогда
         Если МетаРеквизит.Периодический>0 Тогда
           // Если реквизит периодический — нужно установить дату,
           // на которую следует выбрать значение
           sDate = ДАТА(ДатаДляПериод);
           // Проверяем параметр (дату) на корректность
           Если ДатаГод(sDate)<=0 Тогда
             // Дата не корректна — будем ориентироваться
             // на "текущую дату"
             sDate = ТекущаяДата();
           КонецЕсли;
           // Устанавливаем дату выбора значения
           // периодического реквизита
           Спр.ИспользоватьДату(sDate,1);
         КонецЕсли;
         // Читаем значение реквизита из справочника
         V = Спр.ПолучитьАтрибут(""+идРеквизита);
         // Определяем тип значения
         Res = ТипЗначенияСтр(V);
         Если СтрДлина(Res)<=0 Тогда
           Res="Неопределенный";
         КонецЕсли;
         Если ЧИСЛО(ВозвращатьВид)>0 Тогда
           // Нужно вернуть "Вид" значения,
           // если значение имеет агрегатный тип
           ТипЧ = ТипЗначения(V);
           Если ТипЧ>3 Тогда
             // Вычисляем "Вид" значения агрегатного типа
             Res = Res+"."+СокрЛП(V.Вид());
```

```
            КонецЕсли;
          КонецЕсли;
        КонецЕсли;
      КонецЕсли;
    КонецЕсли;
  КонецЕсли;
  Возврат Res;
КонецФункции // гл9_ТипЗначенияСтр_ДляЗначРеквизитаСправочника
```

Посмотрим, как все это работает. Скомпилируем проект и запустим его на выполнение (рис. 4.237).

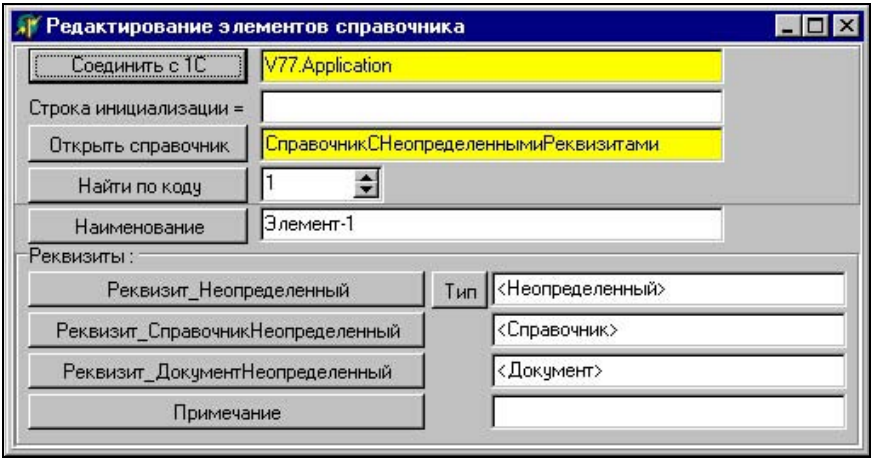



**Рис. 4.237.** Визуализация элемента справочника `СправочникСНеопределеннымиРеквизитами`

Нажмем кнопку **Тип** и введем новый тип `Дата` (рис. 4.238).

На рис. 4.239 представлен результат визуализации элемента справочника после применения метода `НазначитьТип` для реквизита `Реквизит_Неопределенный`.

Если теперь открыть справочник `СправочникСНеопределеннымиРеквизитами` в программе 1С:Предприятие, то реквизит `Реквизит_Неопределенный` для элемента справочника с кодом 1 будет доступен для редактирования.

Запустим нашу программу sp_EditAttrInRef_2.EXE и для разных элементов справочника назначим реквизиту `Реквизит_Неопределенный` разные типы (табл. 4.15).

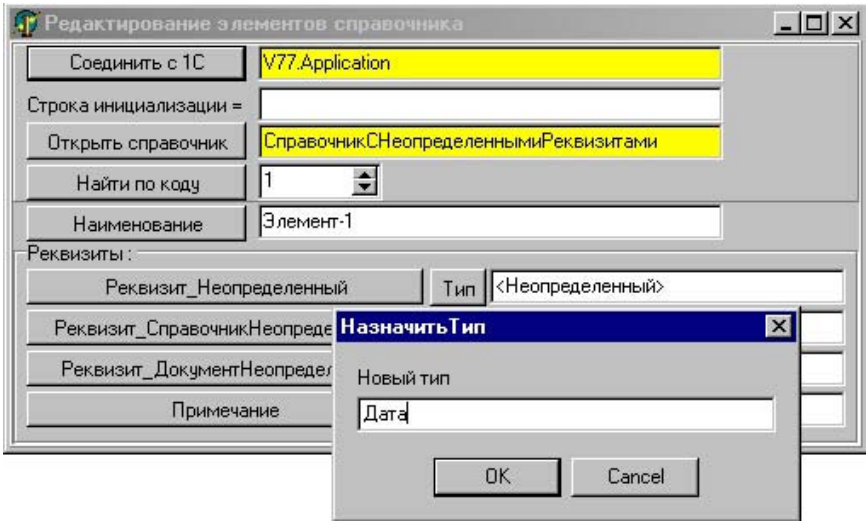



**Рис. 4.238.** Назначение нового типа значению реквизита неопределенного типа

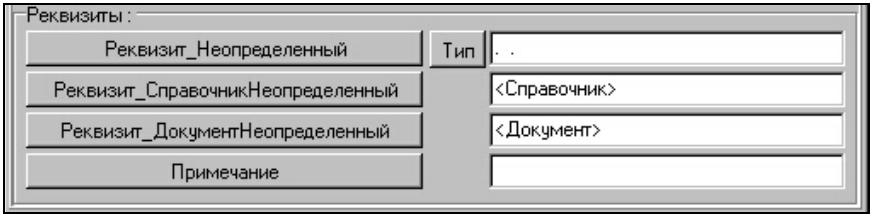



**Рис. 4.239.** Визуализация элемента справочника `СправочникСНеопределеннымиРеквизитами` после изменения типа реквизита `Реквизит_Неопределенный`

***Таблица 4.15.*** *Тип реквизита* `Реквизит_Неопределенный` *для разных элементов справочника*

| Код элемента справочника | Тип |
|---|---|
| 1 | `Число` (длина 10, точность 2) |
| 2 | `Строка` (длина 25) |
| 3 | `Дата` |
| 4 | `Справочник.ФизЛица` |
| 5 | `Документ.ВыплатыФизЛицам` |
| 6 | `Перечисление.ВидВыплатФизЛицам` |

Запустим программу 1С:Предприятие, откроем справочник `СправочникСНеопределеннымиРеквизитами` и отредактируем значение реквизита `Реквизит_Неопределенный` следующим образом (рис. 4.240).

| | Код | Наимен... | Реквизит_Неопределенный |
|---|---|---|---|
| | 1 | Элемент-1 | 125.32 |
| | 2 | Элемент-2 | Это строка |
| | 3 | Элемент-3 | 24.09.06 |
| | 4 | Элемент-4 | Иванов Иван Иванович |
| | 5 | Элемент-5 | ВыплатыФизЛицам 2 |
| | 6 | Элемент-6 | Премия |

**Рис. 4.240.** Справочник `СправочникСНеопределеннымиРеквизитами`. Реквизит `Реквизит_Неопределенный`

Вновь запустим нашу программу и, перемещаясь по элементам справочника, убедимся, что "все работает".

Теперь посмотрим, как можно редактировать значения реквизитов неопределенного типа (по метаданным), если конкретным значениям этих реквизитов назначен какой-либо тип. В рамках проекта sp_EditAttrInRef.DPR мы рассмотрели вопросы редактирования реквизитов разного типа. По сути эти наработки (в целом) применимы и здесь.

Откроем наш проект (sp_EditAttrInRef_2.DPR) и создадим обработчик события: "нажатие кнопки **Реквизит_Неопределенный**" (листинг 4.122).

**Листинг 4.122. Обработчик события "нажатие кнопки *Реквизит_Неопределенный*"**

```
procedure TForm1.SpeedButton2Click(Sender: TObject);
Var
  Sx: string;       // Вспомогательная переменная
  sKind: string;    // Вид значения агрегатного типа
  NrType: integer; // Тип значения реквизита
  sEvalExpr: string;  // Параметр для EvalExpr
  oleBuf,
  oleDoc: oleVariant; // Ссылка на объект типа "Документ"
begin
  // Проверяем, существует ли ссылка на справочник
  if not VarIsEmpty(ole_Ref) then begin
     //-----------------------------
     // Определяем текущий тип значения реквизита
```

```
sEvalExpr:='гл9_ТипЗначения_ДляЗначРеквизитаСправочника(';
sEvalExpr:=sEvalExpr+'"'+Edit_RefName.Text+'",';
sEvalExpr:=sEvalExpr+'"'+'Реквизит_Неопределенный'+'",';
sEvalExpr:=sEvalExpr+SE_Code.Text+',';
sEvalExpr:=sEvalExpr+'"",';
sEvalExpr:=sEvalExpr+'0';
sEvalExpr:=sEvalExpr+')';
NrType:=ole_1C.EvalExpr(sEvalExpr)*1;
//-----------------------------
if NrType <= 0 then begin
   ShowMessage('Сначала нужно реквизиту назначить тип');
end;
if (NrType in [1..3]) then begin
   // Базовый тип
   Sx:=Trim(ole_Ref.Реквизит_Неопределенный);
   if InputQuery('Реквизит_Неопределенный','Новое значение',Sx)
     then begin
      Sx:=Trim(Sx);
      // Редактируем значение реквизита
      ole_Ref.Реквизит_Неопределенный:=Sx;
      // Записываем изменения в справочнике в базу данных
      ole_Ref.Записать;
      // Вновь визуализируем элемент справочника
      SpeedButton9.Click;
   end;
end;
if NrType = 10 then begin
   // Перечисление
   sKind:=Trim(ole_Ref.Реквизит_Неопределенный.Вид);
   Sx:=Trim(ole_Ref.Реквизит_Неопределенный.Идентификатор);
   if Dlg_Select_ValueOfEnum.Execute(ole_1C,sKind,Sx) then begin
      // Редактируем значение реквизита
      ole_Ref.Реквизит_Неопределенный:=
                        Dlg_Select_ValueOfEnum.oleValueEnum;
      // Записываем изменения в справочнике в базу данных
      ole_Ref.Записать;
      // Вновь визуализируем элемент справочника
      SpeedButton9.Click;
```

```
      end;
   end;
   if NrType = 11 then begin
      // Справочник
      sKind:=Trim(ole_Ref.Реквизит_Неопределенный.Вид);
      Sx:=Trim(ole_Ref.Реквизит_Неопределенный.Код);
      if Dlg_Select_ElemFromRef.Execute(ole_1C, sKind, Sx) then begin
         // Редактируем значение реквизита
         ole_Ref.Реквизит_Неопределенный:=
                              Dlg_Select_ElemFromRef.oleElemSel;
         // Записываем изменения в справочнике в базу данных
         ole_Ref.Записать;
         // Вновь визуализируем элемент справочника
         SpeedButton9.Click;
      end;
   end;
   if NrType = 12 then begin
      // Документ
      sKind:=Trim(ole_Ref.Реквизит_Неопределенный.Вид);
      oleDoc:=ole_1C.CreateObject('Документ.'+sKind);
      if (oleDoc.Выбрать('Выберите документ')*1)>0 then begin
         // Читаем ссылку на текущий документ
         oleBuf:=oleDoc.ТекущийДокумент;
         // Еще раз назначаем тип
         // (без этого работает некорректно)
         ole_Ref.НазначитьТип('Реквизит_Неопределенный',
                   'Документ.'+sKind, 0,0);
         //-------------------------------------
         // Редактируем значение реквизита.
         // Такое присвоение не всегда работает корректно:
         // ole_Ref.Реквизит_Неопределенный:=oleDoc.ТекущийДокумент;
         // Тогда можно использовать такой вариант:
         ole_Ref.Реквизит_Неопределенный:=oleBuf;
         //-------------------------------------
         // Записываем изменения в справочнике в базу данных
         ole_Ref.Записать;
         // Вновь визуализируем элемент справочника
         SpeedButton9.Click;
```

```
        end;
      end;
  end;
end;
```

Поясним текст обработчика.

Сначала вызываем функцию `гл9_ТипЗначения_ДляЗначРеквизитаСправочника` в контексте метода `EvalExpr`. Эта функция возвращает в переменной `NrType` код типа *значения* реквизита.

Далее, в зависимости от типа значения реквизита, вызываются различные способы редактирования значения реквизитов (все они, кроме редактирования значений реквизитов типа `Документ`, были подробно рассмотрены ранее). Редактирование значений типа `Документ` нужно рассмотреть более детально.

Как видите, код типа значения `Документ` равен 12. Сначала мы определяем `Вид` (идентификатор) документа:

```
sKind:=Trim(ole_Ref.Реквизит_Неопределенный.Вид);
```

Затем создаем объект типа `Документ` заданного вида:

```
oleDoc:=ole_1C.CreateObject('Документ.'+sKind);
```

С помощью метода `Выбрать` документа (как объекта) выбираем документ (в качестве нового значения для реквизита `Реквизит_Неопределенный`):

```
if (oleDoc.Выбрать('Выберите документ')*1)>0 then begin
```

Получаем ссылку на выбранный документ:

```
oleBuf:=oleDoc.ТекущийДокумент;
```

Затем вновь назначаем соответствующий тип значению реквизита справочника (хотя он был уже назначен ранее):

```
ole_Ref.НазначитьТип('Реквизит_Неопределенный','Документ.'+sKind, 0,0);
```

Без этого тоже почему-то присвоение работает некорректно.

И, наконец-то, присваиваем реквизиту новое значение:

```
ole_Ref.Реквизит_Неопределенный:=oleBuf;
```

Изменения в элементе справочника, как обычно, записываем в базу данных.

```
ole_Ref.Записать;
```

Скомпилируем наш проект, запустим его на выполнение и отредактируем значения реквизита `Реквизит_Неопределенный` для всех элементов справочника `СправочникСНеопределенныминРеквизитами` (рис. 4.241).

Откроем справочник `СправочникСНеопределенныминРеквизитами` в программе 1С:Предприятие и убедимся, что все, что мы напрограммировали, работает (рис. 4.242).

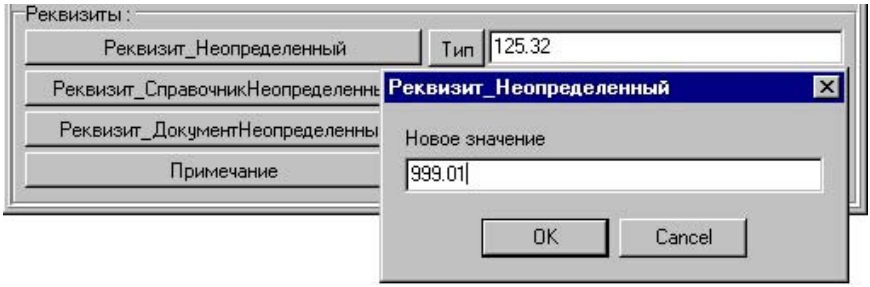



**Рис. 4.241.** Редактирование значения реквизита `Реквизит_Неопределенный`

| | Код | Наименование | Реквизит_Неопределенный |
|---|---|---|---|
| | 1 | Элемент-1 | 999.01 |
| | 2 | Элемент-2 | Это строка - 2 |
| | 3 | Элемент-3 | 28.09.06 |
| | 4 | Элемент-4 | Криогенова Фекла Маринадовна |
| | 5 | Элемент-5 | ВыплатыФизЛицам 3 |
| | 6 | Элемент-6 | МатПомощь |

**Рис. 4.242.** Реквизит `Реквизит_Неопределенный` после редактирования в "нашей" программе

Теперь посмотрим, как можно редактировать значение реквизита типа `Справочник` неопределенного вида (`Реквизит_СправочникНеопределенный` в справочнике `СправочникСНеопределенными Реквизитами`). Такой реквизит позволяет хранить значения, которые могут быть элементами справочников любого типа. Для того чтобы указать, элемент какого именно справочника следует хранить в данном реквизите, нужно вызвать функцию `НазначитьВид` программы 1С:Предприятие. Синтаксис:

`НазначитьВид(`*<Значение>*`, `*<Вид>*`)`

Параметры: *<Значение>* (тип `Документ/Справочник/Счет неопределенного вида`) — обычно реквизит документа, справочника или диалога формы; *<Вид>* — тип `Строка` (вид значения может быть задан строкой, содержащей идентификатор конкретного справочника, документа, плана счетов) или `Вид субконто` (вид значения может быть задан видом субконто, который имеет тип соответственно конкретного справочника, документа, счета).

Процедура `НазначитьВид` (или `SetKind`) позволяет установить конкретный вид для значений типа `Документ неопределенного вида`, `Справочник неопределенного вида`, `Счет неопределенного вида`. При этом, если существующее значение не соответствует вновь установленному виду, оно очищается.

Данная процедура используется обычно для значений реквизитов в диалогах форм. В качестве первого параметра передается значение типа `Справочник`,

Документ или Счет, в котором нужно установить вид. Во втором параметре указывается вид в качестве строки — идентификатора или вида субконто.

Данную процедуру можно использовать в сочетании с методом НеИзменять-Вид элемента формы. Это возволяет регулировать программно собственно вид, а для самого значения предоставить пользователю возможность интерактивного выбора.

> ***Внимание!***
>
> Естественно, что перед вызовом функции НазначитьВид ссылка на справочник должна быть корректной, и справочник должен быть спозиционирован на элементе, к реквизитам которого применяется метод НазначитьВид.

Здесь следует отметить один момент: после вызова функции НазначитьВид нужно сразу же присвоить реквизиту соответствующее значение и вызвать метод справочника Записать.

В листинге 4.123 приведен текст обработчика события "нажатие кнопки **Реквизит_СправочникНеопределенный**" (редактирование значения реквизита элемента справочника).

**Листинг 4.123. Обработчик события "нажатие кнопки *Реквизит_СправочникНеопределенный*"**

```
procedure TForm1.SpeedButton3Click(Sender: TObject);
// Редактирование реквизита "Реквизит_СправочникНеопределенный"
Var
  Sx: string;    // Вспомогательная переменная
  sKind: string; // Вид значения агрегатного типа
begin
  // Проверяем, существует ли ссылка на справочник
  if not VarIsEmpty(ole_Ref) then begin
     Sx:='';
     // Читаем "Вид" значения реквизита
     sKind:=Trim(ole_Ref.Реквизит_СправочникНеопределенный.Вид);
     if length(sKind)>0 then begin
        // Если вид реквизита указан, значит значение корректно
        // и можно прочитать его код (как элемента справочника)
        Sx:=Trim(ole_Ref.Реквизит_СправочникНеопределенный.Код);
     end;
     if InputQuery('Реквизит_СправочникНеопределенный','Новый Вид',sKind)
       then begin
```

```
        sKind:=Trim(sKind);
        // Вызываем функцию "НазначитьВид()" для данного реквизита
ole_1C.НазначитьВид(ole_Ref.Реквизит_СправочникНеопределенный,''+sKind);
        // Выбираем элемент справочника (как новое значение
        // для реквизита "Реквизит_СправочникНеопределенный")
        if Dlg_Select_ElemFromRef.Execute(ole_1C, sKind, Sx) then begin
           // Редактируем значение реквизита
           ole_Ref.Реквизит_СправочникНеопределенный:=
                                   Dlg_Select_ElemFromRef.oleElemSel;
           // Записываем изменения в справочнике в базу данных
           ole_Ref.Записать;
           // Вновь визуализируем элемент справочника
           SpeedButton9.Click;
        end;
     end;
  end;
end;
```

Поясним текст обработчика.

В данном случае мы точно знаем тип значения реквизита (агрегатный тип). Следовательно, значение этого реквизита имеет свойство `Вид` (идентификатор справочника). Мы его считываем в строке:

```
sKind:=Trim(ole_Ref.Реквизит_СправочникНеопределенный.Вид);
```

Запрашиваем новый вид справочника для данного реквизита (данного элемента данного справочника):

```
if InputQuery('Реквизит_СправочникНеопределенный','Новый Вид',sKind)
  then begin
```

Вызываем метод справочника `НазначитьВид` для данного реквизита:

```
ole_1C.НазначитьВид(ole_Ref.Реквизит_СправочникНеопределенный,''+sKind);
```

Далее вызываем диалоговое окно для выбора нового значения реквизита (из множества элементов справочника заданного вида):

```
if Dlg_Select_ElemFromRef.Execute(ole_1C, sKind, Sx) then begin
```

Редактируем значение реквизита:

```
ole_Ref.Реквизит_СправочникНеопределенный:=
                                   Dlg_Select_ElemFromRef.oleElemSel;
```

Изменения в элементе справочника записываем в базу данных:

```
ole_Ref.Записать;
```

Скомпилируем наш проект и запустим его на выполнение.

Проведем тестирование кода (листинг 4.123) на примере элемента (с кодом 0) справочника `СправочникСНеопределенныМиРеквизитами`.

Назначим значению реквизита `Реквизит_СправочникНеопределенный` новый вид `ФизЛица` (рис. 4.243).

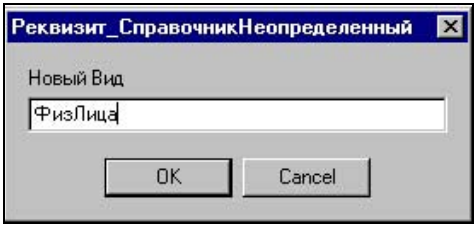



**Рис. 4.243.** Новый вид для реквизита `Реквизит_СправочникНеопределенный`

Затем выберем новое значение для реквизита `Реквизит_СправочникНеопределенный` из множества допустимых значений (элементов справочника `ФизЛица`) — рис. 4.244.

Результат представлен на рис. 4.245.

| Код | Наименование |
|---|---|
| 7 | Группа-1 |
| 8 | Группа-2 |
| 9 | Группа-3 |
| 6 | Криогенова Фекла Маринадовна |

**Рис. 4.244.** Выбор нового значения
для реквизита `Реквизит_СправочникНеопределенный`

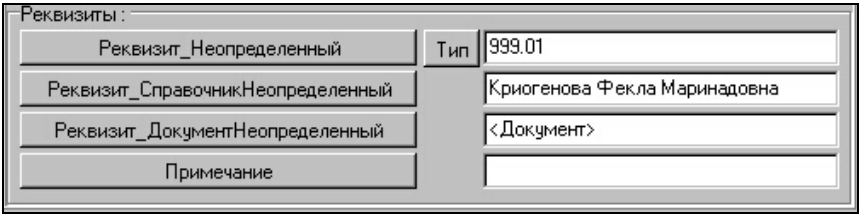



**Рис. 4.245.** Новое значение присвоено реквизиту
`Реквизит_СправочникНеопределенный`

Теперь для этого же элемента справочника назначим этому же реквизиту другой вид (идентификатор справочника) — рис. 4.246.

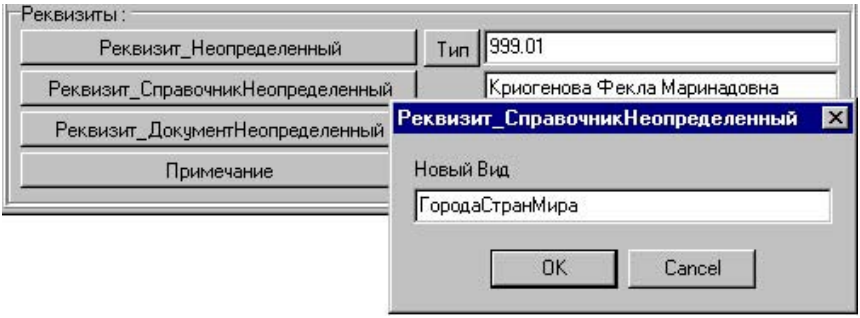



**Рис. 4.246.** Новый вид для реквизита `Реквизит_СправочникНеопределенный`

И выберем новое значение из множества допустимых значений (элементов справочника `ГородаСтранМира`) — рис. 4.247.

Результат представлен на рис. 4.248.

| Код | Наименование |
|---|---|
| 3 | Ильичевск |
| 2 | Киев |
| 1 | Минск |
| 2 | Москва |
| 1 | Одесса |
| 1 | Санкт-Петербург |
| 3 | Усинск |

**Рис. 4.247.** Выбор нового значения для реквизита `Реквизит_СправочникНеопределенный`

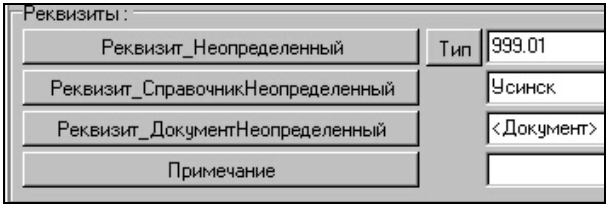



**Рис. 4.248.** Новое значение присвоено реквизиту `Реквизит_СправочникНеопределенный`

Думаю, что здесь комментарии излишни.

Точно такой же подход можно применить и в отношении реквизита типа `Документ` неопределенного вида.

В листинге 4.124 приведен текст обработчика события "нажатие кнопки **Реквизит_ДокументНеопределенный**" (редактирование значения реквизита элемента справочника).

**Листинг 4.124. Обработчик события "нажатие кнопки *Реквизит_ДокументНеопределенный*"**

```
procedure TForm1.SpeedButton4Click(Sender: TObject);
// Редактирование реквизита "Реквизит_ДокументНеопределенный"
Var
 sKind: string; // Вид значения агрегатного типа
 oleBuf: oleVariant; // Вспомогательная переменная
 oleDoc: oleVariant; // Ссылка на объект типа "Документ"
begin
 // Проверяем, существует ли ссылка на справочник
 if not VarIsEmpty(ole_Ref) then begin
   // Читаем "Вид" значения реквизита
   sKind:=Trim(ole_Ref.Реквизит_ДокументНеопределенный.Вид);
   if InputQuery('Реквизит_ДокументНеопределенный',
                 'Новый Вид',sKind) then begin
     sKind:=Trim(sKind);
     // Вызываем функцию "НазначитьВид()" для данного реквизита
ole_1C.НазначитьВид(ole_Ref.Реквизит_ДокументНеопределенный,''+sKind);
     // Создаем объет типа "Документ"
     // и получаем ссылку на него
     oleDoc:=ole_1C.CreateObject('Документ.'+sKind);
     // Выбираем документ (как новое значение для реквизита
     // "Реквизит_ДокументНеопределенный")
     if (oleDoc.Выбрать('Выберите документ')*1)>0 then begin
       // Читаем ссылку на текущий документ
       oleBuf:=oleDoc.ТекущийДокумент;
    //---------------------------------------
    // Редактируем значение реквизита.
    // Это не всегда работает корректно:
    // ole_Ref.Реквизит_ДокументНеопределенный:=oleDoc.ТекущийДокумент;
    // Тогда можно попробовать так:
       ole_Ref.Реквизит_ДокументНеопределенный:=oleBuf;
       //---------------------------------------
       // Записываем изменения в справочнике в базу данных
       ole_Ref.Записать;
```

```
        // Вновь визуализируем элемент справочника
        SpeedButton9.Click;
      end;
    end;
  end;
end;
```

Скомпилируем наш проект и запустим его на выполнение. Проведем тестирование кода (листинг 4.124) на примере элемента (с кодом 0) справочника `СправочникСНеопределеннымиРеквизитами`.

Назначим значению реквизита `Реквизит_ДокументНеопределенный` новый вид `ИзменениеСостоянияФизЛиц` (рис. 4.249).

Выбираем новое значение из множества допустимых значений (документов вида `ИзменениеСостоянияФизЛиц`) — рис. 4.250.

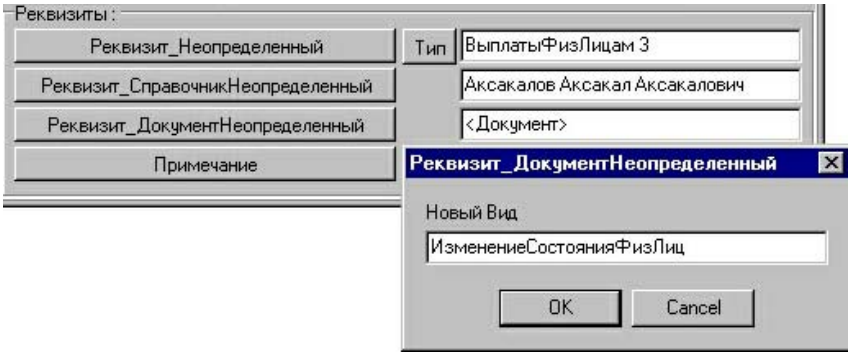



**Рис. 4.249.** Новый вид для реквизита `Реквизит_ДокументНеопределенный`

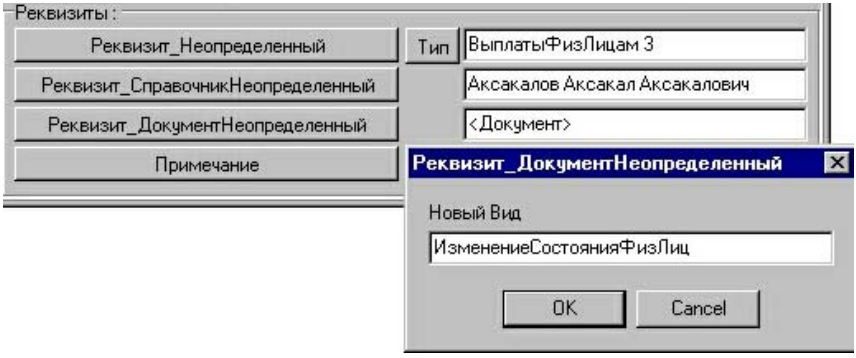



**Рис. 4.250.** Выбор нового значения для реквизита `Реквизит_ДокументНеопределенный`

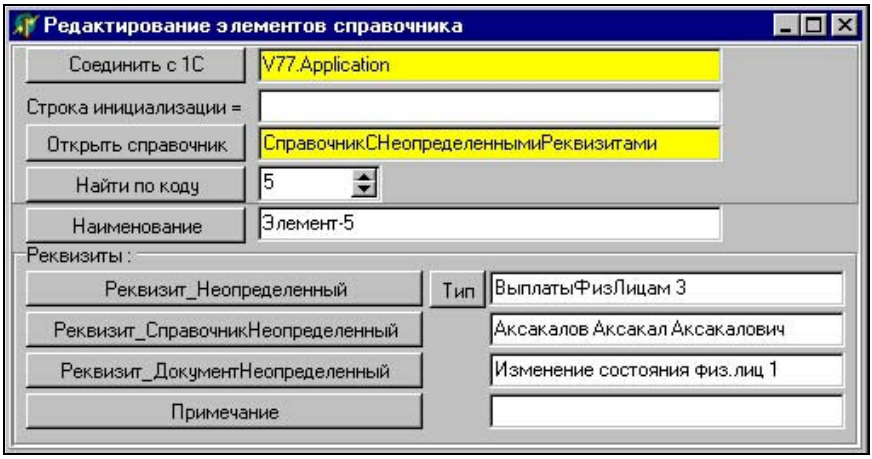



**Рис. 4.251**. Новое значение присвоено реквизиту `Реквизит_ДокументНеопределенный`

Результат представлен на рис. 4.251.

До настоящего момента мы редактировали значения реквизитов справочников, обращаясь к реквизиту непосредственно:

```
ole_Ref.ДатаРождения:=Sx;
ole_Ref.ДополнительнаяИнформация:= Dlg_Edit_AttrString0.Memo1.Text;
ole_Ref.СемейноеПоложение:=Dlg_Select_ValueOfEnum.oleValueEnum;
ole_Ref.ГородПроживания:=Dlg_Select_ElemFromRef.oleElemSel;
ole_Ref.Реквизит_Неопределенный:=Sx;
ole_Ref.Реквизит_Неопределенный:=Dlg_Select_ValueOfEnum.oleValueEnum;
ole_Ref.Реквизит_СправочникНеопределенный:=
                                    Dlg_Select_ElemFromRef.oleElemSel;
ole_Ref.Реквизит_ДокументНеопределенный:=oleBuf;
```

Это приемлемо в том случае, если мы точно знаем, какие именно реквизиты есть в справочнике, и свои программные средства жестко привязываем к информационной структуре конкретной базы данных 1С:Предприятие. В общем случае такой подход будет некорректным.

Объект типа `Справочник` позволяет изменять значения реквизитов элементов справочников (с помощью метода `УстановитьАтрибут`), обращаясь к ним через идентификаторы реквизитов.

Метод `УстановитьАтрибут` (или `SetAttrib`) позволяет установить значение реквизита по идентификатору, как оно задано в Конфигураторе. Синтаксис:

`УстановитьАтрибут(`*<ИмяРеквизита>*`, `*<Значение>*`)`

Параметры: *<ИмяРеквизита>* — строковое выражение, содержащее имя реквизита, как оно задано в Конфигураторе; *<Значение>* — выражение, содержащее устанавливаемое значение реквизита.

Посмотрим, каким образом мы можем применить этот метод при доступе к значениям реквизитов элементов справочников через механизм OLE Automation. Вернемся к нашему проекту sp_EditAttrInRef.DPR и модифицируем его исходный код.

> ***Примечание***
>
> Пример есть на сопроводительном компакт-диске книги в папке \Part_1\Chapter_04\Topic_4_7\Item_4_7_1\sp_EditAttrInRef.PAS\02.

Изменим обработчик события "нажатие кнопки **ДатаРождения**" следующим образом (листинг 4.125).

**Листинг 4.125. Обработчик события "нажатие кнопки *ДатаРождения*"**

```
procedure TForm1.SpeedButton2Click(Sender: TObject);
// Редактирование реквизита "ДатаРождения"
Var
  Sx: string;
begin
  // Проверяем, существует ли ссылка на справочник
  if not VarIsEmpty(ole_Ref) then begin
     Sx:=Edit_DR.Text;
     // Вводим новое значение
     if InputQuery((Sender as TSpeedButton).Caption,'Новое значение',Sx)
       then begin
        Sx:=Trim(Sx);
        // Редактируем значение реквизита
        ole_Ref.УстановитьАтрибут(''+'ДатаРождения',''+Sx);
        // Записываем изменения в справочнике в базу данных
        ole_Ref.Записать;
        // Вновь читаем значение реквизита из справочника
        Edit_DR.Text:=Trim(ole_Ref.ПолучитьАтрибут(''+'ДатаРождения'));
     end;
  end
  else begin
     ShowMessage('Справочник не открыт');
  end;
end;
```

Поясним текст обработчика. Здесь практически то же самое, что и в "старом" варианте (листинг 4.110), кроме двух строк. Строка

```
ole_Ref.ДатаРождения:= Sx;
```

заменена строкой

```
ole_Ref.УстановитьАтрибут(''+'ДатаРождения','+Sx);
```

а строка

```
Edit_DR.Text:=Trim(ole_Ref.ДатаРождения);
```

строкой

```
Edit_DR.Text:=Trim(ole_Ref.ПолучитьАтрибут(''+'ДатаРождения'));
```

Точно так же редактируются реквизиты других базовых типов (`Строка`, `Число`). Пример редактирования реквизита `ДополнительнаяИнформация` (строка неограниченной длины) приведен в листинге 4.126.

**Листинг 4.126. Редактирование реквизита типа `Строка неограниченной длины`**

```
...
Sx:=Dlg_Edit_AttrString0.Memo1.Text;
ole_Ref.УстановитьАтрибут(''+'ДополнительнаяИнформация','+Sx);
...
Sx:=Trim(ole_Ref.ПолучитьАтрибут(''+'ДополнительнаяИнформация'));
Memo1.Text:=Sx;
...
```

Реквизиты агрегатного типа следует рассмотреть подробнее. В листинге 4.127 представлен обработчик события "нажатие кнопки **СемейноеПоложение**" (редактирование реквизита типа `Перечисление`).

**Листинг 4.127. Редактирование реквизита типа `Перечисление`**

```
procedure TForm1.SpeedButton5Click(Sender: TObject);
// Редактирование реквизита "СемейноеПоложение"
Var
  idAttr: string;// Идентификатор реквизита
begin
  // Проверяем, существует ли ссылка на справочник
  if not VarIsEmpty(ole_Ref) then begin
     // Вызываем диалог для выбора значения перечисления
     if Dlg_Select_ValueOfEnum.Execute(ole_1C,
```

```
                                'СемейноеПоложение',
                                Edit_Fam.Text) then begin
        // Редактируем значение реквизита
        idAttr:='СемейноеПоложение';
        ole_Ref.УстановитьАтрибут(''+idAttr,
                                  Dlg_Select_ValueOfEnum.oleValueEnum);
        // Записываем изменения в справочнике в базу данных
        ole_Ref.Записать;
        // Вновь читаем значение реквизита из справочника
        Edit_Fam.Text:=
               Trim(ole_Ref.ПолучитьАтрибут(''+idAttr).Идентификатор);
     end;
  end
  else begin
     ShowMessage('Справочник не открыт');
  end;
end;
```

Этот текст был подробно рассмотрен ранее (см. листинг 4.123). Единственное отличие — использование методов `УстановитьАтрибут` и `ПолучитьАтрибут` справочника.

Здесь следует отметить только один момент. В некоторых случаях передача переменной типа `oleVariant` (или `Variant`) в качестве параметра для метода `УстановитьАтрибут` может не сработать. Тогда можно использовать другой вариант. Среди штатных средств языка программирования системы 1С:Предприятие есть функции `ЗначениеВСтроку` и `ЗначениеИзСтроки`.

Функция `ЗначениеВСтроку` (или `ValueToString`) преобразует значение из внутреннего представления в строковое. Синтаксис:

`ЗначениеВСтроку(<Объект>)`

Параметр: *<Объект>* — значение, которое требуется перевести в строковое представление. В качестве параметра данной функции допускается передавать значения как базовых, так и агрегатных типов данных, за исключением тех типов данных, которые не имеют собственно значения (это касается таких объектов, как `Периодический`, `Таблица`, `ФС`, `Запрос`, `XBase` и т. п.). Возвращаемое значение — строковое представление переданного значения.

Функция `ЗначениеИзСтроки` (или `ValueFromString`) выполняет обратное преобразование из строкового системного представления во внутреннее значение объекта. Синтаксис:

`ЗначениеИзСтроки(<СистСтрока>)`

Параметр: *<СистСтрока>* — строковое выражение, содержащее представление объекта, полученное ранее при помощи функции `ЗначениеВСтроку`. Функция возвращает значение, полученное из строкового системного представления.

> ***Примечание***
>
> Более подробную информацию о функциях `ЗначениеВСтроку` и `ЗначениеИзСтроки` и методах работы с ними можно найти в специальной литературе по 1С.

Обе эти функции предназначены, в частности, для использования при обмене информацией посредством механизма OLE Automation.

Попробуем применить эти функции в контексте метода `EvalExpr`.

В листинге 4.128 представлен модифицированный текст (из предыдущего листинга), где реализован этот способ редактирования реквизитов справочника типа `Перечисление`.

**Листинг 4.128. Редактирование реквизита типа `Перечисление` (`ЗначениеИзСтроки`)**

```
procedure TForm1.SpeedButton5Click(Sender: TObject);
// Редактирование реквизита "СемейноеПоложение"
Var
  Sx,S1,S2: string;// Для метода EvalExpr
  idEnum: string;  // Идентификатор перечисления (вид)
  idAttr: string;  // Идентификатор реквизита
begin
  // Проверяем, существует ли ссылка на справочник
  if not VarIsEmpty(ole_Ref) then begin
     // Идентификатор (вид) перечисления
     idEnum:= 'СемейноеПоложение';
     // Вызываем диалог для выбора значения перечисления
     if Dlg_Select_ValueOfEnum.Execute(ole_1C,idEnum,Edit_Fam.Text)
       then begin
        // Редактируем значение реквизита
        idAttr:='СемейноеПоложение';
        //=========================================
        // Вариант с использованием метода EvalExpr
        //-----------------------------------------
        // Формируем строку для выполнения с помощью метода EvalExpr()
        S1:= Dlg_Select_ValueOfEnum.idValueEnum;
```

```
        Sx:='Перечисление.ПолучитьАтрибут("'+idEnum+'")';
        Sx:=Sx+'.ЗначениеПоИдентификатору("'+S1+'")';
        Sx:='ЗначениеИзСтроки(ЗначениеВСтроку('+Sx+'))';
        //-------------------------------------------
        // Изменяем значение реквизита с помощью метода EvalExpr()
        ole_Ref.УстановитьАтрибут(''+idAttr,ole_1C.EvalExpr(Sx));
        //===========================================
        // Записываем изменения в справочнике в базу данных
        ole_Ref.Записать;
        // Вновь читаем значение реквизита из справочника
        Edit_Fam.Text:=
              Trim(ole_Ref.ПолучитьАтрибут(''+idAttr).Идентификатор);
    end;
  end
  else begin
    ShowMessage('Справочник не открыт');
  end;
end;
```

Здесь дополнительные пояснения излишни.

Сделаем небольшое дополнение в исходном тексте (листинг 4.128):

```
//-------------------------------------------
// Формируем строку для выполнения с помощью метода EvalExpr()
S1:= Dlg_Select_ValueOfEnum.idValueEnum;
Sx:='Перечисление.ПолучитьАтрибут("'+idEnum+'")';
Sx:=Sx+'.ЗначениеПоИдентификатору("'+S1+'")';
Sx:='ЗначениеВСтроку('+Sx+')';
Sx:=Trim(ole_1C.EvalExpr(Sx));
InputQuery('ЗначениеВСтроку','Значение перечисления',Sx);
//-------------------------------------------
```

Теперь можно посмотреть результат, который возвращает метод `ЗначениеВСтроку` для значения реквизита типа `Перечисление`. В данном случае, например, функция вернула следующее значение:

```
{"Перечисление","СемейноеПоложение","ГражданскийБрак"}
```

Если тип редактируемого реквизита — `Справочник`, то можно использовать такой же подход (функцию `ЗначениеИзСтроки`), как и в случае с типом `Перечисление`. Но здесь есть нюанс. Дело в том, что метод `EvalExpr` может вычислить выражение, но не выполнить набор команд. В случае с перечислением

можно было скомбинировать вызов методов (для вычисления нового значения реквизита справочника) таким образом, чтобы все они представляли единое выражение. Для реквизитов типа `Справочник` это невозможно. Но подходящий вариант, как всегда, найдется.

Если провести некоторые эксперименты с функцией `ЗначениеВСтроку` применительно к элементам справочника, то получается вот что.

1. Если справочник *не* иерархический и у него нет владельца, то строковый эквивалент элемента справочника включает в себя 3 слова:
   - первое слово — тип объекта (в данном случае всегда будет `Справочник`);
   - второе слово — идентификатор справочника (вид);
   - третье слово — код выбранного элемента справочника.

   **Пример 1:**

   Справочник: `СтраныМира`.

   Код элемента: 1.

   Строковое представление: `{"Справочник","СтраныМира","1"}`.

2. Если справочник иерархический и у него нет владельца, то строковый эквивалент элемента справочника включает в себя 3 слова:
   - первое слово — тип объекта (в данном случае всегда будет `Справочник`);
   - второе слово — идентификатор справочника (вид);
   - третье слово:
     ◊ если у элемента справочника есть родитель, то формат следующий: `КодРодителя/КодЭлемента` (разделитель — косая черта);
     ◊ если у элемента справочника нет родителя, то просто `КодЭлемента`.

   **Пример 2:**

   Справочник: `ФизЛица`.

   Код элемента: 6.

   Код родителя: нет родителя.

   Строковое представление: `{"Справочник","ФизЛица","6"}`.

   **Пример 3:**

   Справочник: `ФизЛица`.

   Код элемента: 1.

Код родителя: 7.

Строковое представление: `{"Справочник","ФизЛица","7/1"}`.

3. Если справочник иерархический и у него есть владелец, то строковый эквивалент элемента справочника включает в себя 4 слова:
   - первое слово — тип объекта (в данном случае всегда будет `Справочник`);
   - второе слово — идентификатор справочника (вид);
   - третье слово:
     ◊ если у элемента справочника есть родитель, то формат следующий: `КодРодителя/КодЭлемента` (разделитель — косая черта);
     ◊ если у элемента справочника нет родителя, то просто `КодЭлемента`.
   - четвертое слово — код владельца выбранного элемента справочника.

**Пример 4:**

Справочник: `ГородаСтранМира`.

Код элемента: 3.

Код родителя: нет родителя.

Код владельца: 1.

Строковое представление: `{"Справочник","ГородаСтранМира","3","1"}`.

**Пример 5:**

Справочник: `Новый2_Подч`.

Код элемента: 3.

Код родителя: 1.

Код владельца: 2.

Строковое представление: `{"Справочник","Новый2_Подч","1/3","2"}`.

То есть формат строки прост, и не составляет труда сформировать такую строку средствами Delphi (не обращаясь к функции 1С `ЗначениеВСтроку`), чтобы потом передать эту строку в качестве параметра в функцию `ЗначениеИзСтроки`.

Текст функции (Delphi), решающей эту задачу, представлен в листинге 4.129.

**Листинг 4.129. Функция `sp_ValueToString_Ref` (Delphi) — аналог функции `ЗначениеВСтроку` (1С:Предприятие)**

```
function sp_ValueToString_Ref(handle1C,
                              oleCurItemRef: oleVariant): string;
// Аналог функции "ЗначениеВСтроку" программы 1С:Предприятие
```

```
// (для элементов справочника):
// handle1C — ссылка на 1С,
// oleCurItemRef — ссылка на элемент справочника
//                 (который конвертируем в строку).
//==============================
// Формат результирующей строки:
// {"Справочник","ИдентСправочника","КодРодителя/КодЭлемента",
//    "КодВладельца"}
//==============================
Var
  idRef: string; // Идентификатор справочника
                 // (для значения oleCurItemRef)
  CodeCurItemRef: string; // Код элемента справочника oleCurItemRef
  CodeParentCurItemRef: string; // Код элемента справочника-родителя
                                // для oleCurItemRef
  CodeOwnerCurItemRef: string;  // Код элемента справочника-владельца
                                // для oleCurItemRef
  oleBuf: oleVariant; // Вспомогательная переменная
  idType: string;     // Тип значения oleCurItemRef
begin
  Result:='';
  // Тип значения (в данном случе — всегда "Справочник")
  idType:='Справочник';
  CodeOwnerCurItemRef:='';
  CodeParentCurItemRef:='';
  // Проверяем, есть ли ссылка на 1С
  if not VarIsEmpty(handle1C) then begin
     // Проверяем, корректна ли ссылка на элемент справочника
     if not VarIsEmpty(oleCurItemRef) then begin
        // Идентификатор справочника
        idRef:=Trim(oleCurItemRef.Вид);
        // Код элемента справочника oleCurItemRef
        CodeCurItemRef:=Trim(oleCurItemRef.Код);
        // Проверяем, есть ли владелец у этого справочника
        oleBuf:=handle1C.Метаданные.Справочник(''+idRef).Владелец;
        if oleBuf.Выбран>0 then begin
           // Владелец есть
           // Код элемента справочника-владельца
```

```
   CodeOwnerCurItemRef:=Trim(oleCurItemRef.Владелец.Код);
end;
// Проверяем, является ли справочник иерархическим
oleBuf:=handle1C.Метаданные.Справочник(''+idRef);
if (oleBuf.КоличествоУровней*1)>1 then begin
   // Справочник иерархический
   // Код элемента справочника-родителя
   CodeParentCurItemRef:=Trim(oleCurItemRef.Родитель.Код);
end;
//-----------------------------------
// Формируем результат.
// Двойные кавычки нужно дублировать, потому что
// в 1С:Предприятие в двойные кавычки заключаются
// строковые константы.
if length(CodeParentCurItemRef)<=0 then begin
   // Справочник НЕ иерархический
   if length(CodeOwnerCurItemRef)>0 then begin
      // Есть владелец
      Result:='{""'+idType+'"","+'""'+idRef+'"","""
                  +CodeCurItemRef+'"","""
                  +CodeOwnerCurItemRef+'""}';
   end
   else begin
      // Нет владельца
      Result:='{""'+idType+'"","+'""'
                  +idRef+'"","""+CodeCurItemRef+'""}';
   end;
end
else begin
   // Справочник иерархический
   if length(CodeOwnerCurItemRef)>0 then begin
      //Есть владелец
      Result:='{""'+idType+'"","+'""'+idRef+'"","""
            +CodeParentCurItemRef+'/'+CodeCurItemRef+'"","""
            +CodeOwnerCurItemRef+'""}';
   end
   else begin
```

```
            // Нет владельца
            Result:='{""'+idType+'"","+'""'+idRef+'"","""'
                   +CodeParentCurItemRef+'/'+CodeCurItemRef+'""}';
         end;
      end;
      //-----------------------------------
    end;
  end;
end;
```

Текст функции `sp_ValueToString_Ref` прост и не требует дополнительных пояснений. Применить ее можно так, как показано в листинге 4.130.

**Листинг 4.130. Применение функции `sp_ValueToString_Ref`**

```
 ...
// Вызываем диалог для выбора элемента справочника
 if Dlg_Select_ElemFromRef.Execute(ole_1C,
                            'ГородаСтранМира', '') then begin
    // Идентификатор реквизита справочника
    idAttr:= 'ГородПроживания';
    // Вариант с использованием метода EvalExpr
    //-------------------------------------------
    // Формируем строковое представление выбранного
    // элемента справочника (новое значение для реквизита)
    Sx:=sp_ValueToString_Ref(ole_1C, Dlg_Select_ElemFromRef.oleElemSel);
    //-------------------------------------------
    // Изменяем значение реквизита с помощью метода EvalExpr()
    ole_Ref.УстановитьАтрибут(''+idAttr,
                   ole_1C.EvalExpr('ЗначениеИзСтроки("'+Sx+'")'));
    // Записываем изменения в справочнике в базу данных
    ole_Ref.Записать;
    ...
end;
```

Другой вариант применения метода `УстановитьАтрибут` для реквизитов типа `Справочник` (как и в случае с реквизитами типа `Перечисление`):

```
ole_Ref.УстановитьАтрибут(''+idAttr, Dlg_Select_ElemFromRef.oleElemSel);
```

Использование метода `УстановитьАтрибут` для значений реквизитов типа: `Неопределенный`, `Справочник` (неопределенного вида), `Документ` (неопределенного вида) ничем не отличается от уже рассмотренных примеров. Но перед вызовом этого метода нужно назначить соответствующий тип (вид) значению такого реквизита справочника (этот вопрос был подробно рассмотрен ранее).

***Примечание***

В папке \Part_1\Chapter_04\Topic_4_7\Item_4_7_1\sp_EditAttrInRef_2.PAS\04 на сопроводительном компакт-диске книги есть исходные тексты проекта sp_EditAttrInRef_2.DPR, иллюстрирующие этот вопрос.

Теперь рассмотрим вопросы, связанные с редактированием значений периодических реквизитов. Откроем Конфигуратор 1С:Предприятие и создадим новый справочник, который содержит только периодические реквизиты разных типов (рис. 4.252).

**Рис. 4.252.** Новый справочник для иллюстрации вопросов, связанных с редактированием периодических реквизитов справочников

В табл. 4.16 приведена информация о типе периодических реквизитов справочника `СправПериодРекв`.

***Таблица 4.16.*** *Типы реквизитов справочника* `СправПериодРекв`

| Реквизит | Тип | Вид |
|---|---|---|
| Тип_Строка | Строка | — |
| Тип_Число | Число | — |
| Тип_Дата | Дата | — |
| Тип_Справ_ФизЛица | Справочник | ФизЛица |
| Тип_Переч_СемейноеПоложение | Перечисление | СемейноеПоложение |
| Тип_Док_ИзмСостФизЛица | Документ | ИзменениеСостоянияФизЛиц |
| Тип_Неопределенный | Неопределенный | Неопределенный |
| Тип_Справочник | Справочник | Неопределенный |
| Тип_Документ | Документ | Неопределенный |

Как было сказано в *разд. 4.1.5*, периодические реквизиты — это реквизиты, значения которых связаны с датой. При изменении значения периодического реквизита старое значение сохраняется, при этом новое значение действует с указанной даты, а старое — до указанной даты.

**ПРИМЕЧАНИЕ**

Более подробно о перидических реквизитах и методах работы с ними можно прочитать в специальной литературе по 1С.

Здесь мы рассмотрим вопросы, связанные с редактированием таких реквизитов, на конкретных примерах.

Внести новое значение периодического реквизита справочника можно с помощью метода `УстановитьАтрибут` справочника (см. ранее в этом разделе), но перед его вызовом нужно вызвать метод `ИспользоватьДату` справочника *(см. разд. 4.1.5)*.

Для иллюстрации использования этих методов создадим новый Delphi-проект sp_EditAttrPeriodInRef.DPR.

**ПРИМЕЧАНИЕ**

Исходные тексты проекта находятся на сопроводительном компакт-диске книги в папке \Part_1\Chapter_04\Topic_4_7\Item_4_7_1\sp_EditAttrPeriodInRef.PAS\01.

На рис. 4.253 представлена форма проекта sp_EditAttrPeriodInRef.DPR.

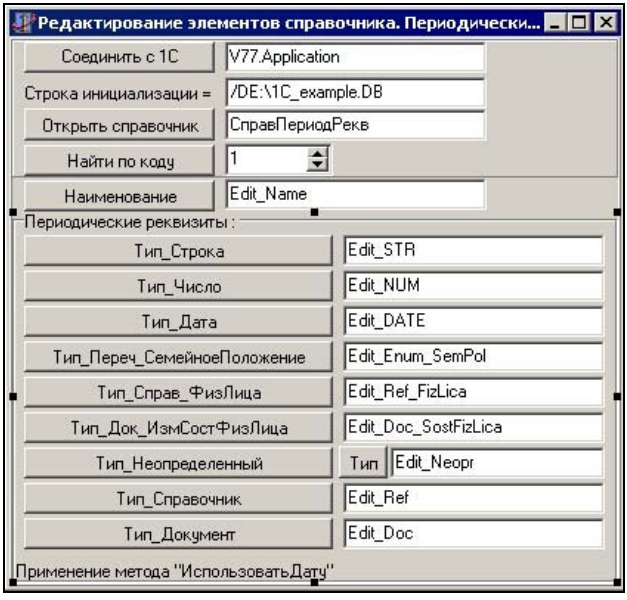



**Рис. 4.253.** Форма проекта sp_EditAttrPeriodInRef.DPR для иллюстрации процесса редактирования значений периодических реквизитов справочника

Как видите, для каждого периодического реквизита справочника `СправПериодРекв`, создана отдельная кнопка. В листинге 4.131 приведен пример обработчика события: "нажатие кнопки **Тип_Дата**".

**Листинг 4.131. Обработчик события "нажатие кнопки *Тип_Дата*"**

```
procedure TForm1.SpeedButton4Click(Sender: TObject);
// Редактирование периодического реквизита "Тип_Дата"
Var
  Sx: string;
  idAttr: string; // Идентификатор реквизита
begin
  // Проверяем, существует ли ссылка на справочник
  if not VarIsEmpty(ole_Ref) then begin
     // Перед вводом нового значения периодического реквизита
     // с применением метода "УстановитьАтрибут" нужно вызвать
     // метод "ИспользоватьДату" для установки даты значения
     // реквизита
     ole_Ref.ИспользоватьДату(''+DateToStr(NOW),1);
     idAttr:='Тип_Дата';
     Sx:=Edit_DATE.Text;
     // Вводим новое значение
     if InputQuery(idAttr,'Новое значение на дату='+DateToStr(NOW),Sx)
       then begin
        Sx:=Trim(Sx);
        // Редактируем значение реквизита
        ole_Ref.УстановитьАтрибут(''+idAttr,''+Sx);
        // Записываем изменения в справочнике в базу данных
        ole_Ref.Записать;
        // Вновь читаем значения реквизитов из справочника
        SpeedButton9.Click;
     end;
  end
  else begin
     ShowMessage('Справочник не открыт');
  end;
end;
```

Если сравнить листинги 4.131 (редактирование периодического реквизита `ДатаРождения`) и 4.125 (редактирование *не* периодического реквизита `ДатаРождения`), то видно, что единственное существенное отличие — вызов метода справочника `ИспользоватьДату` перед вызовом `УстановитьАтрибут`.

Результат представлен на рис. 4.254.

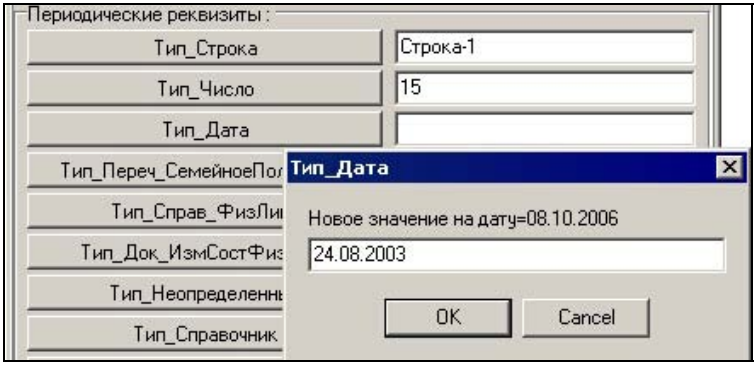



**Рис. 4.254.** Новое значение периодического реквизита `Тип_Дата` справочника `СправПериодРекв` на 08.10.2006

Такой же подход (сочетание методов `ИспользоватьДату` и `УстановитьАтрибут`) можно применять для периодических реквизитов других типов. В случае, если периодический реквизит имеет тип `Неопределенный`, `Справочник неопределенного вида` или `Документ неопределенного вида`, то дополнительно вызываются методы `НазначитьТип` и `НазначитьВид`, как и для непериодических реквизитов (см. ранее).

Например, обращение к методу `НазначитьВид` может быть следующим:

```
idAttr:= 'Тип_Справочник';// Идентификатор реквизита
sKind:= 'ГородаСтранМира';
ole_1C.НазначитьВид(ole_Ref.ПолучитьАтрибут(''+idAttr),''+sKind);
```

Ничего особенного здесь нет, поэтому не будем заострять на этом внимание. Желающие могут посмотреть исходные тексты проекта sp_EditAttrPeriodInRef.DPR.

Единственное, что следует отметить, — для заданной даты может быть только одно значение периодического реквизита (созданное без участия документов). А это значит, что если в базе данных уже существует значение периодического реквизита с заданной датой (не связанное с документом), то при вызове метода `ИспользоватьДату` (с этой же датой) метод `УстановитьЗначение` модифицирует существующее значение периодического реквизита. Если же

в базе данных нет значения (не связанного с документом) с этой датой, то будет добавлено новое значение с заданной датой.

На рис. 4.255 представлен результат ввода новых значений периодических реквизитов для элемента (с кодом 1) справочника `СправПериодРекв`.

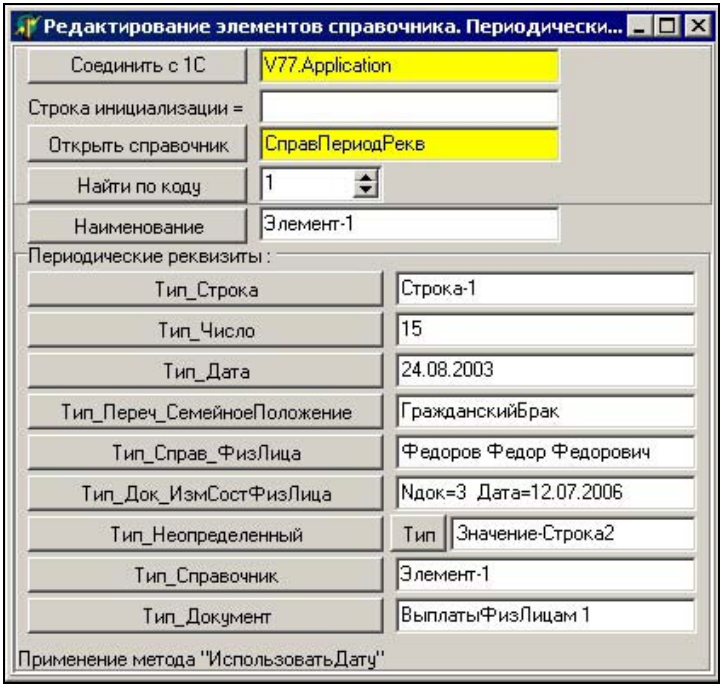



**Рис. 4.255.** Результат ввода новых значений периодических реквизитов элемента (с кодом 1) справочника `СправПериодРекв`

Как отмечалось в *разд. 4.1.5*, для работы с периодическими реквизитами справочников и периодическими константами в системе используется специальный агрегатный тип данных — `Периодический`. Объекты данного типа предоставляют возможность записи, редактирования и удаления значений периодических реквизитов справочников и периодических констант (т. е. позволяют работать с историей значений периодического реквизита). Модифицируем наш проект sp_EditAttrPeriodInRef.DPR следующим образом (рис. 4.256).

Как видите, для каждого периодического реквизита добавлена новая кнопка **История**.

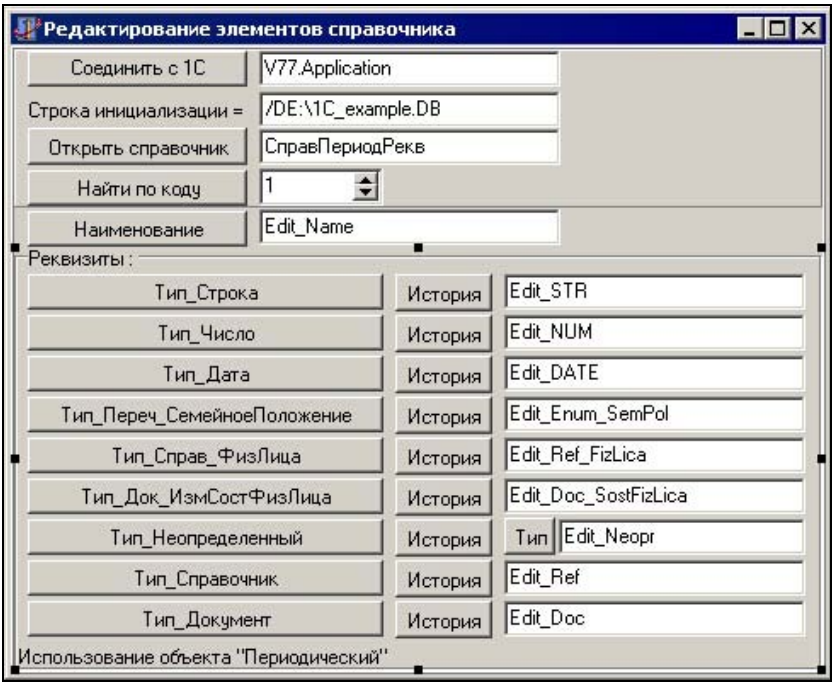



**Рис. 4.256.** Модифицированная форма проекта sp_EditAttrPeriodInRef.DPR

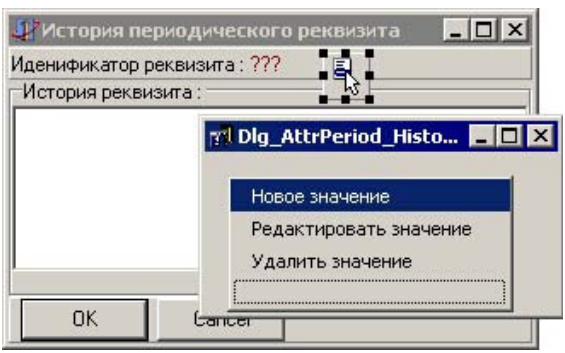



**Рис. 4.257.** Модифицированная форма (Delphi) для визуализации и редактирования истории изменения значений периодического реквизита

### *Примечание*

Исходные тексты этого примера находятся на сопроводительном компакт-диске книги в папке \Part_1\Chapter_04\Topic_4_7\Item_4_7_1\sp_EditAttrPeriodInRef.PAS\02.

Добавим к проекту форму, разработанную нами в *разд. 4.1.5* (см. рис. 4.89), и модифицируем ее, добавив контекстное меню (рис. 4.257).

В листинге 4.132 представлен текст обработчика события "выбор пункта меню **Новое значение**".

**Листинг 4.132. Добавление нового значения периодического реквизита**

```
procedure TDlg_AttrPeriod_History.N1Click(Sender: TObject);
// Новое значение периодического реквизита
Var
  NrType: integer; // Тип реквизита по метаданным
  Sx: string;
  sDate,sValAttr: string;
  ole_period: OleVariant; // Ссылка на объект "Периодический"
begin
 // Поскольку значения периодических реквизитов
 // рассматриваются в контексте конкретного элемента
 // справочника, ссылка на текущий элемент справочника
 // ole_TekEl должна быть корректной.
 if not VarIsEmpty(ole_TekEl) then begin
   ole_period:=UnAssigned;
   sDate:=DateTostr(NOW);
   // Вводим дату нового значения реквизита
   if InputQuery('Периодический реквизит',
                 'Введите дату нового значения',
                 sDate) then begin
     sDate:=Trim(sDate);
     if length(sDate)>0 then begin
       // Создаем объект "Периодический" для того,
       // чтобы прочитать историю значений периодического
       // реквизита справочника
       ole_period:=Form1.ole_1C.CreateObject('Периодический');
       if not VarIsEmpty(ole_period) then begin
         // Устанавливаем фильтр по текущему элементу справочника
         if ole_period.ИспользоватьОбъект(''+id_AttrPeriod,
                                          ole_TekEl)*1>0 then begin
           // Определяем тип реквизита по метаданным
           NrType:=GetTypeAttrReference_1C(Form1.ole_1C,
                                           Label_id_Ref.Caption,
                                           id_AttrPeriod);
```

```
          if (NrType in [1..3]) then begin
            // Реквизит имеет базовый тип
            // Вводим новое значение реквизита
            sValAttr:='';
            if InputQuery('Периодический реквизит',
                          'Введите новое значение',
                          sValAttr) then begin
              sValAttr:=Trim(sValAttr);
              //----------------------------
              // Присваиваем новое значение и дату
              // периодическому реквизиту
              ole_period.ДатаЗнач:= sDate;
              ole_period.Значение:= sValAttr;
              //----------------------------
              //----------------------------
              // Записываем в базу данных (здесь следует помнить,
              // что все значения периодических реквизитов хранятся
              // в отдельной таблице 1SCONST.DBF.
              // Очевидно, что поэтому у объекта "Периодический"
              // есть свой метод "Записать")
              ole_period.Записать;
              //----------------------------
              // Визуализируем историю значений реквизита
              View_Values;
            end;
          end;
        end;
      end;
    end;
  end;
  ole_period:=UnAssigned;
 end;
end;
```

Думаю, дополнительно пояснять текст этого обработчика не требуется. Единственное, что следует отметить, — метод `Записать` объекта `Периодический` выполняет запись значения периодического реквизита на заданную дату. Как уже отмечалось, если для этой даты уже имеется запись (созданная без участия документов), то она модифицируется (если новое значение тоже

не связано с каким-то документом). А это значит, что этот же обработчик можно использовать и для *редактирования* значений периодического реквизита.

Следует помнить, что при редактировании конкретного значения реквизита объект `Периодический` должен быть позиционирован на этом значении. Для этого можно использовать метод `НайтиЗначение` (или `FindValue`), позволяющий найти значение на заданную дату. Его синтаксис:

`НайтиЗначение(<Дата>, <Режим>)`

Параметры: *<Дата>* — выражение, задающее значение даты, на которую требуется найти периодический реквизит справочника или периодическую константу; *<Режим>* — числовое выражение, значение которого задает режим поиска, если на заданную дату не существует значения периодического реквизита (–1 — возвращается значение на предыдущую дату, 0 — возвращается код завершения неуспешной операции, 1 — возвращается значение на следующую дату). Возвращаемое значение (число): 1 — если вызов метода закончился успешно, 0 — если нет.

Метод `НайтиЗначение` позволяет найти периодическое значение на заданную дату *<Дата>*. Режим поиска в случае, если на заданную дату не существует значения периодического реквизита, задается параметром *<Режим>*. Само полученное значение следует считывать из атрибута `Значение`.

В нашем случае обращение к этому методу может быть таким:

```
if (ole_period.НайтиЗначение(''+sDate,0)*1) > 0 then begin
```

Мы рассмотрели возможность применения объекта `Периодический` для редактирования значений периодических реквизитов на примере реквизита базового типа. С помощью наших предыдущих наработок можно по такому же принципу реализовать ввод новых и редактирование существующих значений реквизитов агрегатного типа (включая реквизиты неопределенного вида).

Для удаления конкретного значения периодического реквизита служит метод `Удалить` объекта `Периодический`. При этом объект `Периодический` должен быть позиционирован на удаляемом значении. Это можно сделать, используя (например) его метод `НайтиЗначение`, который позволяет позиционировать объект `Периодический` на значении по заданной дате.

Желающие ознакомиться подробнее могут посмотреть исходные тексты проекта, а также почитать специальную литературу по этому вопросу.

### Примечание

Напомню, что представленные выше примеры исходных текстов — это упрощенная иллюстрация к рассматриваемым вопросам. Назначение этих примеров — дать представление о возможном подходе при решении задач информационного обмена с системой 1С:Предприятие. При работе с реальными базами 1С может потребоваться их дальнейшая доработка.

## 4.7.2. Редактирование документов

Как отмечалось в *разд. 4.1.5*, реквизиты документов разделяются на:

- общие реквизиты документов;
- реквизиты "шапки" документов конкретного вида;
- реквизиты "табличной части" документов конкретного вида.

Каждый реквизит документа заданного вида имеет уникальный идентификатор. Причем уникальность соблюдается среди всего множества реквизитов для документов конкретного вида и множества общих реквизитов документов. То есть если среди общих реквизитов документов есть реквизит с идентификатором `Комментарий`, то никакие документы (в рамках данной конфигурации) любого вида не могут иметь реквизиты с таким же идентификатором ни в "шапке" документов, ни в "табличной части" документов. Точно так же, если в "шапке" документа (конкретного вида) есть реквизит с каким-то идентификатором, то среди реквизитов "табличной части" документов этого же вида не может быть реквизита с таким же индетификатором. Это существенно упрощает как считывание информации из документов *(см. разд. 4.2)*, так и редактирование информации в документах.

Рассмотрим вопросы редактирования информации в документах с использованием механизма OLE Automation на примере документов вида `ВыплатыФизЛицам` (см. рис. 4.166). Для этого создадим новый проект sp_EditAttrInDoc.DPR.

> ***Примечание***
>
> Исходные тексты проекта есть на сопроводительном компакт-диске книги в папке \Part_1\Chapter_04\Topic_4_7\Item_4_7_2\sp_EditAttrInDoc.PAS.

На рис. 4.258 представлена форма проекта.

Форма проекта разделена на три части.

- Верхняя часть — соединение с 1С, с документами вида `ВыплатыФизЛицам`, навигация (позиционирование) по списку документов данного вида.
- Средняя часть — общие реквизиты документов и реквизиты "шапки" документов данного вида.
- Нижняя часть — реквизиты "табличной части" документов данного вида.

Вопросы, связанные с открытием документов и позиционированием на конкретном документе заданного вида, были подробно рассмотрены в *разд. 4.2*, и здесь мы на этом останавливаться не будем.

**Рис. 4.258.** Форма проекта sp_EditAttrInDoc.DPR

Посмотрим, каким образом можно редактировать общие реквизиты документов и реквизиты "шапки" документов. В листинге 4.133 представлен обработчик события "нажатие кнопки **Комментарий**" (общий реквизит документов).

**Листинг 4.133. Редактирование общих реквизитов документов**

```
procedure TForm1.SpeedButton3Click(Sender: TObject);
// Редактирование реквизита "Комментарий"
Var
  idAttr: string;   // Идентификатор реквизита документа
  NumLine: integer; // Номер текущей строки
  Sx: string;       // Вспомогательная переменная
begin
  // Проверяем, существует ли ссылка на документ
  if not VarIsEmpty(ole_Doc) then begin
     // Убеждаемся, что объект ole_Doc позиционирован
     // на конкретном документе
```

```
      //(т. е. ссылка на текущий документ корректна)
      if not VarIsEmpty(ole_TekDoc) then begin
         // Идентификатор реквизита
         idAttr:='Комментарий';
         Sx:=Trim(Edit_Comment.Text);
         // Вызываем диалог для ввода строки
         if InputQuery(idAttr, 'Введите новое значение',Sx) then begin
            Sx:=Trim(Sx);
            // Редактируем значение реквизита
            ole_Doc.УстановитьАтрибут(''+idAttr,''+Sx);
            // Записываем изменения в документе в базу данных
            ole_Doc.Записать;
            // Это нужно, чтобы после повторного открытия документа
            // вернуться на текущую строку
            NumLine:=SE_NumLine.Value;
            // Визуализируем документ
            SpeedButton9.Click;
            // Возвращаемся на текущую строку документа
            SE_NumLine.Value:=NumLine;
         end;
      end
      else begin
         ShowMessage('Такой документ не существует в базе данных');
      end;
   end
   else begin
      ShowMessage('Документ не открыт');
   end;
end;
```

Поясним текст обработчика.

Здесь проверяется корректность ссылки на текущий документ (это значит, что созданный объект `ole_Doc` позиционирован на конкретном документе):

```
if not VarIsEmpty(ole_TekDoc) then begin
```

Далее следует запрос на ввод нового значения реквизита:

```
if InputQuery(idAttr, 'Введите новое значение',Sx) then begin
```

Затем модифицируем значение реквизита:

```
ole_Doc.УстановитьАтрибут(''+idAttr,''+Sx);
```

И записываем изменения в базу данных:

```
ole_Doc.Записать;
```

Точно такой же подход можно использовать и при редактировании реквизитов "шапки" документов (листинг 4.134).

**Листинг 4.134. Редактирование реквизитов "шапки" документов**

```
procedure TForm1.SpeedButton2Click(Sender: TObject);
// Редактирование реквизита "Город"
Var
  idAttr: string;   // Идентификатор реквизита документа
  NumLine: integer; // Номер текущей строки
begin
  // Проверяем, существует ли ссылка на документ
  if not VarIsEmpty(ole_Doc) then begin
     // Убеждаемся, что объект ole_Doc позиционирован
     // на конкретном документе
     // (т.е. ссылка на текущий документ корректна)
     if not VarIsEmpty(ole_TekDoc) then begin
       // Идентфикатор реквизита
       idAttr:='Город';
       // Вызываем диалог для выбора элемента справочника
       // (новое значение реквизита документа)
       if Dlg_Select_ElemFromRef.Execute(ole_1C, 'ГородаСтранМира', '')
         then begin
          // Редактируем значение реквизита
          ole_Doc.УстановитьАтрибут(''+idAttr,
                          Dlg_Select_ElemFromRef.oleElemSel);
          // Записываем изменения в документе в базу данных
          ole_Doc.Записать;
          // Это нужно, чтобы после повторного открытия документа
          // вернуться на текущую строку
          NumLine:=SE_NumLine.Value;
          // Визуализируем документ
          SpeedButton9.Click;
          // Возвращаемся на текущую строку документа
          SE_NumLine.Value:=NumLine;
       end;
```

```
      end
      else begin
         ShowMessage('Такой документ не существует в базе данных');
      end;
   end
   else begin
      ShowMessage('Документ не открыт');
   end;
end;
```

Как видите, листинг 4.133 отличается от листинга 4.134 только способом ввода нового значения реквизита документа. В первом случае реквизит `Комментарий` имеет тип `Строка`. Во втором случае реквизит `Город` имеет тип `Справочник.ГородаСтранМира`. Поэтому во втором случае для ввода нового значения мы используем разработанную нами ранее диалоговую форму для выбора элементов справочника заданного вида.

При редактировании реквизитов табличной части документа можно использовать точно такой же подход с учетом того, что данный документ (текущий документ) должен быть позиционирован на конкретной строке "табличной части". Для этого можно использовать метод `ПолучитьСтрокуПоНомеру` (ил `GetLineByNum`) объекта `Документ`, позволяющий получить строку документа по заданному номеру. Синтаксис:

`ПолучитьСтрокуПоНомеру(<Номер>)`

Параметр: `<Номер>` — выражение, содержащее номер искомой строки. Возвращаемое значение (число): 1 — если действие выполнено; 0 — если действие не выполнено.

Метод `ПолучитьСтрокуПоНомеру` устанавливает в качестве текущей строки спецификации документа строку с заданным номером (не порядковым, а тем, который записан в строке, т. к. могут быть пропуски).

В листинге 4.135 представлен исходный текст обработчика, где производится редактирование реквизита "табличной части" документа (кнопка **Способ Выплаты**).

**Листинг 4.135. Редактирование реквизитов "табличной части" документов**

```
procedure TForm1.SpeedButton10Click(Sender: TObject);
// Редактирование реквизита "СпособВыплаты"
Var
  idAttr: string;   // Идентификатор реквизита документа
```

```
  NumLine: integer; // Номер текущей строки
  Yes: integer;     // Вспомогательная переменная
begin
 // Проверяем, существует ли ссылка на документ
 if not VarIsEmpty(ole_Doc) then begin
   // Убеждаемся, что объект ole_Doc позиционирован
   // на конкретном документе
   // (т.е. ссылка на текущий документ корректна)
   if not VarIsEmpty(ole_TekDoc) then begin
     // Убеждаемся, что номер строки - в допустимом интервале
     if SE_NumLine.Value>0 then begin
        if SE_NumLine.Value<=(ole_Doc.КоличествоСтрок*1) then begin
           // Позиционируемся на выбранной строке
           // табличной части документа
           Yes:=(ole_Doc.ПолучитьСтрокуПоНомеру(SE_NumLine.Value*1)*1);
           if Yes>0 then begin
             // Идентификатор реквизита
             idAttr:= 'СпособВыплаты';
             // Вызываем диалог для выбора элемента справочника
             if Dlg_Select_ValueOfEnum.Execute(ole_1C,
                             'СпособВыплатыФизЛицам',
                             Edit_SposVyplat.Text) then begin
               // Редактируем значение реквизита
               ole_Doc.УстановитьАтрибут(''+idAttr,
                           Dlg_Select_ValueOfEnum.oleValueEnum);
               // Записываем изменения в документе в базу данных
               ole_Doc.Записать;
               // Это нужно, чтобы после повторного открытия документа
               // вернуться на текущую строку
               NumLine:=SE_NumLine.Value;
               // Визуализируем документ
               SpeedButton9.Click;
               // Возвращаемся на текущую строку документа
               SE_NumLine.Value:=NumLine;
             end;
           end;
        end;
     end;
```

```
      end;
    end
    else begin
       ShowMessage('Такой документ не существует в базе данных');
    end;
  end
  else begin
     ShowMessage('Документ не открыт');
  end;
end;
```

Как видите, единственное существенное отличие от предыдущих листингов — позиционирование строки документа по номеру:

```
Yes:=(ole_Doc.ПолучитьСтрокуПоНомеру(SE_NumLine.Value*1)*1);
if Yes>0 then begin
...
```

Для удаления строки из табличной части документа служит метод `УдалитьСтроку` (при этом документ должен быть позиционирован на удаляемой строке):

```
if (ole_Doc.ПолучитьСтрокуПоНомеру(SE_NumLine.Value*1)*1)>0 then begin
  ole_Doc.УдалитьСтроку;
  ole_Doc.Записать;
  ...
```

Для удаления всех строк из табличной части документа предназначен метод `УдалитьСтроки`:

```
ole_Doc.УдалитьСтроки;
ole_Doc.Записать;
```

Внести новую строку в табличную часть документа можно с помощью метода `НоваяСтрока`:

```
ole_Doc.НоваяСтрока;
ole_Doc.Записать;
```

Внести новый документ в список документов заданного вида можно с помощью метода `Новый` документа:

```
if not VarIsEmpty(ole_Doc) then begin
  ole_Doc.Новый;
  ole_Doc.Записать;
  ...
```

Для проведения документов заданного вида служит метод `Провести`:

```
if not VarIsEmpty(ole_Doc) then begin
  ole_Doc.Провести(0,'');
end;
```

Скомпилируем наш проект sp_EditAttrInDoc.DPR, запустим его на выполнение и отредактируем реквизиты "шапки" и первой строки табличной части документа `ВыплатыФизЛицам` (с номером 1) от 14.07.2006.

На рис. 4.258 и 4.259 показан этот документ до редактирования.

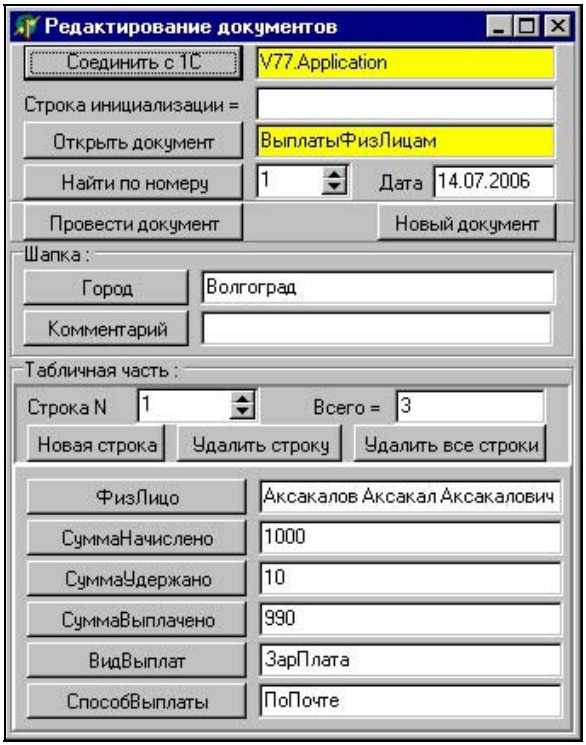



**Рис. 4.259.** Документ `ВыплатыФизЛицам` (с номером 1) от 14.07.2006 до редактирования

А на рис. 4.260 и 4.261 — после редактирования.

Если посмотреть движения этого документа до редактирования (рис. 4.262) и после редактирования, то видно, что ничего не изменилось.

То есть изменения в документе сделаны, но документ не проведен, и поэтому движения по документу (а значит и по регистрам) остались старые.

Шапка :

Город: Москва

Комментарий: В табличной части изменена строка N

Табличная часть :

Строка N: 1 Всего = 3

Новая строка | Удалить строку | Удалить все строки

ФизЛицо: Федоров Федор Федорович

СуммаНачислено: 2000

СуммаУдержано: 50

СуммаВыплачено: 1950

ВидВыплат: Аванс

СпособВыплаты: ЧерезКассу

**Рис. 4.260.** Документ ВыплатыФизЛицам (с номером 1) от 14.07.2006 после редактирования

НомерДок: 1 ДатаДок: 14.07.06

Город: Москва Комментарий: В табличной части изменена стро

| N | ФизЛицо | СуммаНа... | СуммаУдер... | СуммаВы... | ВидВып... | СпособВыплаты |
|---|---|---|---|---|---|---|
| 1 | Федоров Федор Фед | 2000 | 50 | 1950 | Аванс | ЧерезКассу |
| 2 | Аксакалов Аксакал А | 30 | 1 | 29 | МатПомощь | ПоПочте |
| 3 | Сидоров Сидор Сидор | 300 | 3 | 297 | Премия | ЧерезБанк |

**Рис. 4.261.** Документ ВыплатыФизЛицам (с номером 1) от 14.07.2006 после редактирования (просмотр в программе 1С:Предприятие)

| | N... | Город | Физ лицо | ВидВып... | Сумма начис... | СуммаУд... | СуммаВы... | СпособВыпла |
|---|---|---|---|---|---|---|---|---|
| | 1 | Волгогр | Аксакалов Акс | ЗарПлата | 1000 | 10 | 990 | ПоПочте |
| | 2 | Волгогр | Аксакалов Акс | МатПомощь | 30 | 1 | 29 | ПоПочте |
| | 3 | Волгогр | Сидоров Сидор | Премия | 300 | 3 | 297 | ЧерезБанк |

**Рис. 4.262.** Движения по документу ВыплатыФизЛицам (с номером 1) от 14.07.2006 до редактирования (просмотр в программе 1С:Предприятие)

| | N.. | Город | Физ лицо | ВидВып... | Сумма н... | СуммаУд... | СуммаВы... | СпособВыпл |
|---|---|---|---|---|---|---|---|---|
| | 1 | Москва | Федоров Федс | Аванс | 2000 | 50 | 1950 | ЧерезКассу |
| | 2 | Москва | Аксакалов Акс | МатПомощь | 30 | 1 | 29 | ПоПочте |
| | 3 | Москва | Сидоров Сидор | Премия | 300 | 3 | 297 | ЧерезБанк |

**Рис. 4.263.** Движения по документу ВыплатыФизЛицам (с номером 1) от 14.07.2006 после редактирования и проведения (просмотр в программе 1С:Предприятие)

Вновь запустим программу sp_EditAttrInDoc.EXE, проведем этот документ (кнопка **Провести документ**), закроем программу, запустим программу 1С:Предприятие и посмотрим движение по этому документу (рис. 4.263).

Теперь все так, как и должно быть. Движения по документу соответствуют информационному содержанию документа.

***Примечание***

Более подробные сведения о способах редактирования информации в документах можно найти в специальной литературе по программе 1С:Предприятие.

***Примечание***

Напомню, что представленные примеры исходных текстов — это упрощенная иллюстрация к рассматриваемым вопросам. Назначение этих примеров — дать представление о возможном подходе при решении задач информационного обмена с системой 1С:Предприятие. При работе с реальными базами 1С может потребоваться их дальнейшая доработка.

# 4.8. Синхронизация баз данных 1С

В процессе эксплуатации программных продуктов 1С на предприятии может возникнуть необходимость частой и периодической (например, ежечасной) частичной синхронизации разных баз данных (возможно, разных конфигураций). Может оказаться, что использование механизма OLE Automation в этом случае будет наиболее оптимальным.

Один из возможных вариантов синхронизации (применительно к справочникам, например) может быть таким:

1. Заранее определяются справочники и их реквизиты, которые должны быть включены в процесс синхронизации.
2. В случае разных справочников — определяются соответствия реквизитов (по идентификаторам) и правила конвертации (если реквизиты имеют разные типы).
3. Открывается база данных № 1.
4. Считывается информация из метаданных о структуре синхронизируемого справочника А базы данных № 1.
5. Проверяется соответствие списка реквизитов справочника А, заданных для синхронизации, и списка реквизитов, реально существующих в справочнике А (лишние из списка синхронизируемых удаляются).
6. Открывается справочник А базы данных № 1.

7. Открывается база данных № 2.
8. Считывается информация из метаданных о структуре синхронизируемого справочника Б базы данных № 2.
9. Проверяется соответствие списка реквизитов справочника Б, заданных для синхронизации, и списка реквизитов, реально существующих в справочнике Б (лишние из списка синхронизируемых удаляются).
10. Открывается справочник Б базы данных № 2.
11. Сканируется справочник А и поэлементно проверяется (по уникальным кодам или реквизитам) наличие элементов в справочнике Б.
12. Если такого элемента в справочнике Б нет, то добавляется новый элемент и копируются значения синхронизируемых реквизитов (по заранее установленным правилам, если типы реквизитов отличаются).

    Если такой элемент в справочнике Б есть, то значения синхронизируемых реквизитов в справочнике Б заменяются на значения соответствующих реквизитов из справочника А (по заранее установленным правилам, если типы реквизитов отличаются).

В предыдущих разделах мы подробно рассмотрели все то, что потребуется для реализации этого алгоритма, а именно:

- соединение с 1С как сервером OLE Automation;
- работа с метаданными;
- выборка элементов справочников;
- поиск элемента справочника;
- считывание информации из элемента справочника;
- редактирование информации в элементе справочника;
- вычисление значений в строковом представлении (методы `EvalExpr`, `ЗначениеВСтроку`, `ЗначениеИзСтроки`).

Разработка универсальных программных средств, ориентированных на синхронизацию баз данных 1С (с использованием механизма OLE Automation), — существенно трудоемкое занятие. Поэтому мы рассмотрим этот процесс на простом примере синхронизации двух "родственных" справочников в разных конфигурациях.

Для этого скопируем конфигурацию 1С_example.DB в каталог с именем 1С_example.DB_synhro и отредактируем новую конфигурацию таким образом, чтобы структуры справочника `ФизЛица` отличались (рис. 4.264 и 4.265). Будем считать базу данных 1С_example.DB *эталонной*, а базу данных 1С_example.DB_synhro — *синхронизируемой*. Примем также, что элементы

справочников ФизЛица синхронизируются по всем свойствам (Наименование, Код, Родитель) и реквизитам, которые выбирает пользователь (типы реквизитов одинаковы).

**Рис. 4.264.** Структура эталонного справочника ФизЛица базы данных 1С_example.DB

**Рис. 4.265.** Структура синхронизируемого справочника ФизЛица базы данных 1С_example.DB_synhro

Как видите, в целом структура обоих справочников совпадает, за исключением реквизита СостоитВПрофсоюзе.

Теперь запустим программу 1С:Предприятие (конфигурация 1С_example.DB_synhro) и сделаем непроведенными *все* документы вида ИзменениеСостояния-ФизЛиц (рис. 4.266). Это нужно для того, чтобы удалить из справочника ФизЛица значения всех периодических реквизитов, созданные при проведении документов.

Откроем справочник ФизЛица и внесем следующие изменения.

1. Перенесем элемент с кодом 5 (Аксакалов Аксакал Аксакалович) из группы с кодом 7 (Группа-1) в группу с кодом 9 (Группа 3).
2. Очистим историю всех периодических реквизитов (СостояниеФизлица и КолвоДетей) для этого же элемента справочника.
3. Откроем этот же элемент справочника для редактирования и внесем следующие изменения (рис. 4.267).

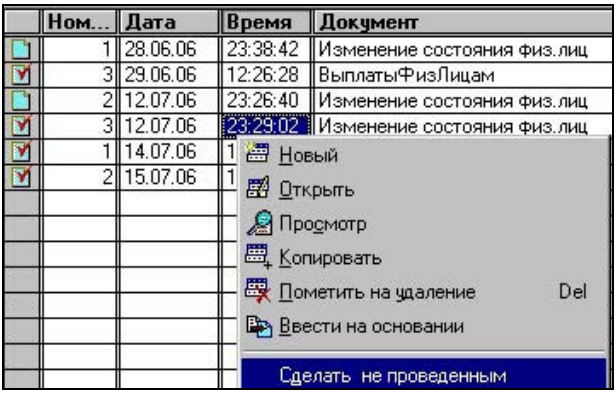



**Рис. 4.266.** Сделаем документ "непроведенным"

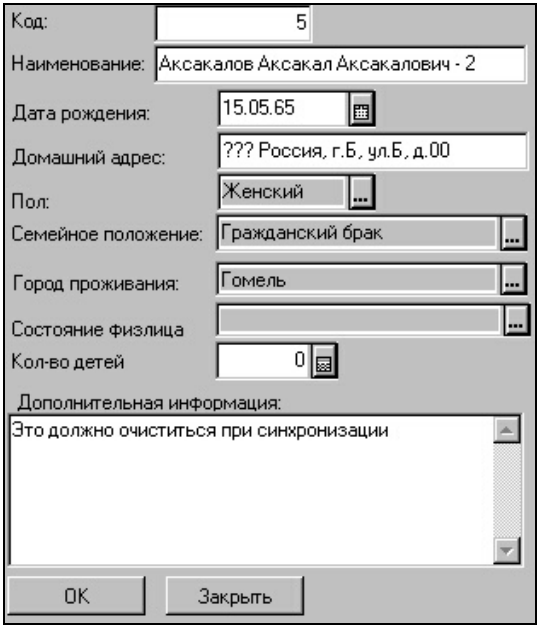



**Рис. 4.267.** Содержание элемента справочника с кодом 5 в базе данных 1C_example.DB_synhro

Закроем программу 1С:Предприятие.

Для сравнения на рис. 4.268 представлено информационное содержание элемента справочника с кодом 5 в эталонной базе данных 1C_example.DB.

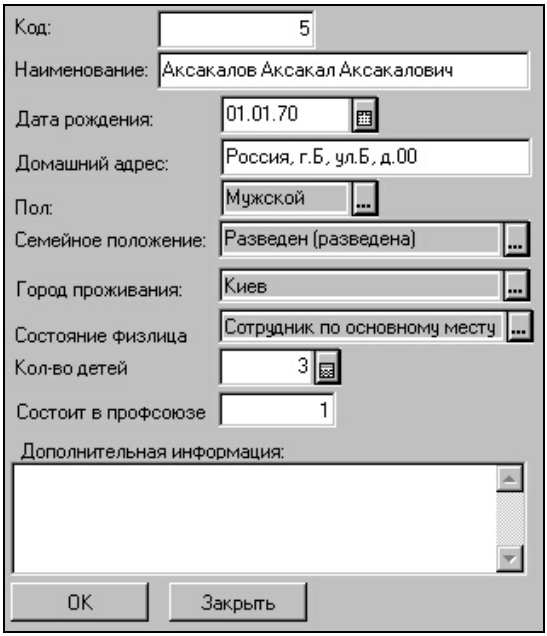



**Рис. 4.268.** Содержание элемента справочника с кодом 5 в базе данных 1C_example.DB

В предыдущих разделах мы начинали рассматривать какой-либо вопрос "с начала", т. е. с программирования. В данном случае поступим с точностью до наоборот. Запустим программу sp_SynhroRefs.EXE, которая находится на сопроводительном компакт-диске книги в папке \Part_1\Chapter_04\Topic_4_8\sp_SynhroRefs.PAS\, и "пощупаем", как она работает. Поочередно нажимаем кнопки **1С** на левой и правой панели (естественно, в строке инициализации должен быть указан реальный путь доступа к конфигурациям). Результат представлен на рис. 4.269.

Нажимаем кнопку **Мета**. Результат представлен на рис. 4.270.

Список, показанный на рис. 4.270, можно отредактировать так, как это нужно для текущей задачи (например, удалить "лишние" реквизиты). Если нажать кнопку **Контроль**, то список реквизитов автоматически "синхронизируется" таким образом, что в списке останутся "общие" реквизиты (по их идентификаторам). В нашем случае так и нужно поступить. После этого закроем эту форму нажатием кнопки **OK**.

Теперь отметим в списке элементов строку с кодом 5 и нажмем кнопку **Спр**. Результат представлен на рис. 4.271. Как видите, информация совпадает с представленной на рис. 4.267 и 4.268.

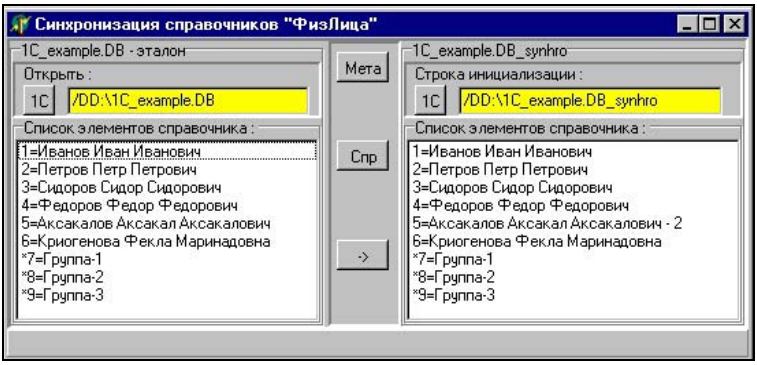



**Рис. 4.269.** Список элементов справочников `ФизЛица` в обеих конфигурациях

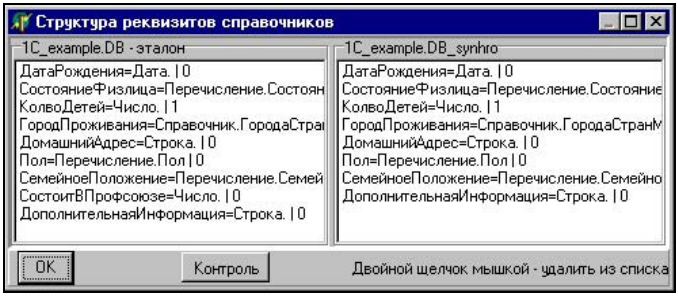



**Рис. 4.270.** Список реквизитов для синхронизации

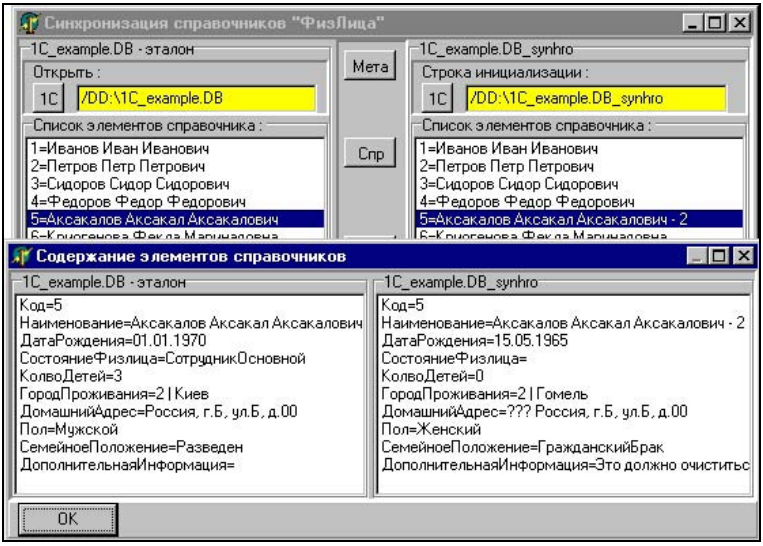



**Рис. 4.271.** Содержание элемента справочника с кодом 5
в обеих базах данных до синхронизации

Закроем это диалоговое окно и нажмем кнопку -> ("синхронизировать"). По окончании синхронизации на экране появится сообщение (рис. 4.272).

Вновь нажмем кнопку **Спр**. Результат представлен на рис. 4.273.

**Рис. 4.272.** Сообщение об окончании синхронизации элемента справочников

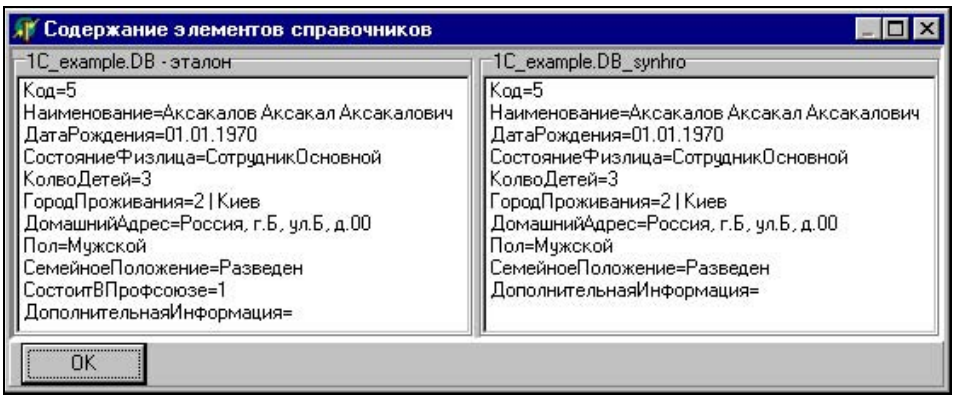



**Рис. 4.273.** Содержание элемента справочника с кодом 5 в обеих базах данных после синхронизации

Комментарии, думаю, здесь не требуются.

Откроем программу 1С:Предприятие (конфигурация 1С_example.DB_synhro) и убедимся, что элемент справочника с кодом 5 из группы "Группа-3" "вернулся в свою alma mater ("Группа-1"). А также посмотрим, что произошло с периодическими реквизитами (для этого же элемента справочника) на примере реквизита `СостояниеФизлица` (рис. 4.274).

| | Дата | Время | Документ | Номер | N С... | Значение |
|---|---|---|---|---|---|---|
| | 07.06.06 | | | | | Временно не работает |
| | 24.06.06 | | | | | Сотрудник по неосновному месту работы |
| | 28.06.06 | | | | | Сотрудник по основному месту работы |
| | 12.07.06 | | | | | Сотрудник по основному месту работы |

**Рис. 4.274.** Значения периодического реквизита `СостояниеФизлица` для элемента справочника с кодом 5 в базе данных 1С_example.DB_synhro после синхронизации

Для сравнения на рис. 4.275 представлены значения этого же реквизита для элемента справочника с кодом 5 в эталонной базе данных 1С_example.DB.

| | Дата | Время | Документ | Номер | N С... | Значение |
|---|---|---|---|---|---|---|
| | 07.06.06 | | | | | Временно не работает |
| | 24.06.06 | | | | | Сотрудник по неосновному месту работы |
| | 28.06.06 | 23:38:42 | Изменение сс | 1 | 1 | Сотрудник по основному месту работы |
| | 12.07.06 | 23:26:40 | Изменение сс | 2 | 2 | Сотрудник по основному месту работы |

**Рис. 4.275.** Значения периодического реквизита `СостояниеФизлица` для элемента справочника с кодом 5 в базе данных 1С_example.DB

Здесь нужны пояснения. Как видите, значения реквизитов и даты совпадают в обоих случая. Но в эталонной базе данных 1С_example.DB (см. рис. 4.275) два последних значения "привязаны" к документам, а в базе данных 1С_example.DB_synhro (см. рис. 4.274) — нет. Это было сделано преднамеренно, поскольку в данном случае мы синхронизировали *только* справочники и не касались документов. А это значит, что списки документов в обеих базах данных могут отличаться. Более того, в случае "разнородных" баз данных в синхронизируемой базе вообще может не быть документов соответствующего вида.

Теперь откроем исходные тексты проекта sp_SynhroRefs.DPR (папка \Part_1\Chapter_04\Topic_4_8\sp_SynhroRefs.PAS\ на сопроводительном компакт-диске книги) и попробуем разобраться, как все это работает.

> ***Примечание***
>
> Вопросы, связанные с запуском 1С (как сервера автоматизации), а также с формированием списка элементов справочников и списка синхронизируемых реквизитов (из метаданных), подробнейшим образом рассмотрены ранее.

Сразу же перейдем к процессу синхронизации выбранного элемента справочника (по нажатию кнопки **->**). Текст обработчика события: "нажатие этой кнопки" представлен в листинге 4.136.

**Листинг 4.136. Обработчик события "нажатия кнопки ->"**

```
procedure TForm1.SpeedButton1Click(Sender: TObject);
// Синхронизация текущего элемента справочника
Var
  Sx: string;
  sCode: string; // Код синхронизируемого элемента справочника
begin
  if ListBox1.ItemIndex>=0 then begin
```

```
      // Если какой-то элемент справочника в списке выбран,
      // считываем текущую строку в списке-эталоне
      //(см. рис. 4.271)
      Sx:=Trim(ListBox1.Items[ListBox1.ItemIndex]);
      // Выделяем код элемента справочника
      sCode:=Trim(sp_GetWordFromString('=', Sx, 1));
      // Вызываем функцию, где реализован алгоритм
      // синхронизации элемента справочника по его коду
      if SynhroThisElem(sCode) then begin
         ShowMessage('Готово');
      end
      else begin
         ShowMessage('Неудачно ...');
      end;
   end;
end;
```

Здесь пояснения не требуются.

Более подробно мы рассмотрим реализацию функции `SynhroThisElem`.

> ***Примечание***
>
> Полный исходный текст этой функции здесь не приводится (из-за экономии места, а также по той причине, что здесь это будет "нечитабельно").

Первое, что "делает" функция, — убирает "звездочку" из строковой переменной, где хранится код элемента (если элемент является группой, в начало строки добавлен символ "звездочка"):

```
// sCode — значение кода элемента, который синхронизируется
sCode:=sp_AllTrimUnChar('* ' , sCode);
```

Затем позиционируется справочник-эталон на выбранном элементе (код которого передан функции в качестве параметра в переменной `sCode`):

```
if (oleRef_FZ_1.НайтиПоКоду(StrToInt(sCode),0)*1) >0 then begin
```

Поскольку справочник `ФизЛица` иерархический, определяется код родителя выбранного элемента-эталона. Если у данного элемента есть родитель, то синхронизируемый справочник позиционируется на этом родителе (по коду) и в переменной `oleParent2` сохраняется ссылка на текущий элемент (соответствующий найденному родителю):

```
SCodeParent:='';
// Проверяем, есть ли у этого элемента родитель
```

```
if (oleRef_FZ_1.Родитель.Выбран*1)>0 then begin
    // У данного элемента справочника есть родитель.
    // Определяем код родителя.
    sCodeParent:=Trim(oleRef_FZ_1.Родитель.Код);
    // Ищем ЭТОГО родителя в синхронизимруемой базе данных
    if (oleRef_FZ_2.НайтиПоКоду(StrToInt(sCodeParent),0)*1) >0 then begin
        oleParent2:= oleRef_FZ_2.ТекущийЭлемент;
```

**Примечание**

Если в справочнике-эталоне родитель существует, а в синхронизируемом — нет, то сначала нужно завести этого родителя в синхронизируемом справочнике (как группу), а затем синхронизировать элемент справочика. Отсюда следует, что если синхронизация справочников выполняется автоматически (сразу по всем элементам), то сначала (на первом проходе) нужно синхронизировать элементы-группы (чтобы создать "отсутствующих" родителей), а на втором проходе — остальные элементы.

После "выявления" родителя синхронизируемый справочник позиционируется на элементе с кодом `sCode` (это тот элемент, который выбран для синхронизации). Если элемент не найден, создаем новый. Если это группа, вызываем метод `НоваяГруппа`, для остальных элементов — метод `Новый`. Затем присваиваем значения свойств `Код` и `Наименование` элемента-эталона соответствующим свойствам созданного элемента (синхронизируемого справочника) и сохраняем созданный элемент справочника в синхронизируемой базе данных:

```
// В синхронизируемом справочнике ищем элемент с кодом sCode
if (oleRef_FZ_2.НайтиПоКоду(StrToInt(sCode),0)*1)<=0 then begin
   // Если элемента в справочнике синхронизируемой базы данных
   // нет, то нужно внести новый элемент справочника
   //------------------------
   if (oleRef_FZ_1.ЭтоГруппа*1)<=0 then begin
      // Это НЕ группа — вызываем метод "Новый"
      oleRef_FZ_2.Новый;
   end
   else begin
      // Это группа — вызываем метод "НоваяГруппа"
      oleRef_FZ_2.НоваяГруппа;
   end;
   //------------------------
   //------------------------
   // Присваиваем такие же код и наименование, как
```

```
   // в элементе справочника эталонной базы данных
   oleRef_FZ_2.Код:=sCode;
   oleRef_FZ_2.Наименование:=oleRef_FZ_1.Наименование;
   //------------------------
   // Записываем новый элемент справочника
   // в базу данных
   oleRef_FZ_2.Записать;
```

Далее вновь позиционируемся на элементе синхронизируемого справочника (следуя логике, это уже лишнее, но с 1С лучше перестраховаться) и редактируем свойства `Наименование` и `Родитель` элемента:

```
if (oleRef_FZ_2.НайтиПоКоду(StrToInt(sCode),0)*1) >0 then begin
   oleRef_FZ_2.Наименование:=oleRef_FZ_1.Наименование;
   //------------------------
   // Привязываем этот элемент справочника
   // к родителю
   if not VarIsEmpty(oleParent2) then begin
      // Родитель существует
      oleRef_FZ_2.Родитель:=oleParent2;
   end
   else begin
      // Родитель НЕ существует
      oleRef_FZ_2.Родитель:='';
   end;
   //------------------------
```

Проверяем: если синхронизируемый элемент — это группа, то записываем изменения в синхронизируемую базу данных (для группы реквизиты не синхронизируем):

```
if (oleRef_FZ_1.ЭтоГруппа*1)>0 then begin
   // Это группа — реквизиты синхронизировать не будем.
   // Записываем изменения в элементе справочника
   // в синхронизируемую базу данных
   oleRef_FZ_2.Записать;
end;
```

Проверяем: если синхронизируемый элемент — это *не* группа, то вызываем функцию `SynhroThisElem_Attr` для синхронизации реквизитов и, если удачно, сохраняем изменения в синхронизируемой базе данных:

```
if (oleRef_FZ_1.ЭтоГруппа*1)<=0 then begin
   // Это НЕ группа.
```

```
   // Вызываем функцию для редактирования реквизитов
   if SynhroThisElem_Attr then begin
      // Все ОК — записываем изменения в элементе справочника
      // в базу данных
      oleRef_FZ_2.Записать;
   end
```

Теперь посмотрим, каким образом можно синхронизировать элементы справочников (функция `SynhroThisElem_Attr`). Сначала определяем — задан ли список реквизитов для синхронизации (была ли нажата кнопка **Мета**, см. рис. 4.270):

```
CountAttr:=Dlg_AttrStr.ListBox1.Items.Count;
if CountAttr>0 then begin
```

Если да, то сканируем список реквизитов (справочника-эталона) и выделяем из списка идентификаторы реквизитов:

```
iAttr:=-1;
while iAttr<(CountAttr-1) do
  begin
    iAttr:=iAttr+1;
    // Читаем идентификатор очередного реквизита
    // справочника эталонной базы данных
    Sx:=Trim(Dlg_AttrStr.ListBox1.Items[iAttr]);
    idAttr:=Trim(sp_GetWordFromString('=', Sx, 1));
```

Затем (на всякий случай) определяем, существует ли реквизит с этим идентификатором в синхронизируемом справочнике:

```
// Получаем ссылку на синхронизируемый реквизит
// как объект метаданных
oleMeta2:=ole1C_2.Метаданные.Справочник(''+'ФизЛица');
oleMeta2:=oleMeta2.Реквизит(''+idAttr);
if oleMeta2.Выбран*1>0 then begin
   // Такой реквизит есть в синхронизируемой базе данных
```

Если такой реквизит имеется в синхронизируемом справочнике, определяем его тип:

```
//---------------------------
// Читаем свойства синхронизируемого реквизита:
// "Периодичность" реквизита
ItsPeriod2:=oleMeta2.Периодический*1;
// Тип реквизита
```

```
sType2:= Trim(oleMeta2.Тип);
// Вид реквизита
sKind2:= Trim(oleMeta2.Вид);
//---------------------------
```

Затем считываем тип реквизита в справочнике-эталоне:

```
// Получаем ссылку на реквизит-эталон как объект метаданных
oleMeta1:=ole1C_1.Метаданные.Справочник(''+'ФизЛица');
oleMeta1:=oleMeta1.Реквизит(''+idAttr);
// Читаем свойства реквизита-эталона:
//"Периодичность" реквизита
ItsPeriod1:=oleMeta1.Периодический*1;
// Тип реквизита
sType1:= Trim(oleMeta1.Тип);
// Вид реквизита
sKind1:= Trim(oleMeta1.Вид);
```

Сравниваем типы реквизитов в обеих конфигурациях. Если типы несовместимы, то должны быть предварительно назначены правила конвертации. В противном случае данный реквизит пропускается (в нашем случае этого делать не надо):

```
if (sType1 = sType2) and
  (sKind1 = sKind2) and
  (ItsPeriod1 = ItsPeriod2) then begin
  // Данный реквизит в обеих базах данных
  // "одинаков" (для наших целей)
```

Теперь проверяем: если реквизит *не* периодический, то читаем значение реквизита с помощью метода `ПолучитьАтрибут`:

```
if ItsPeriod1<=0 then begin
  // Это непериодический реквизит.
  // Читаем значение реквизита-эталона
  oleVal:=oleRef_FZ_1.ПолучитьАтрибут(''+idAttr);
```

И синхронизируем с помощью метода `УстановитьАтрибут` (как мы делали это в *разд. 4.7.1*).

Для реквизитов базового типа:

```
if (AnsiUpperCase(sType1)=AnsiUpperCase('Строка')) or
  (AnsiUpperCase(sType1)=AnsiUpperCase('Число')) or
  (AnsiUpperCase(sType1)=AnsiUpperCase('Дата')) then begin
```

```
  // Это базовый тип.
  // Записываем эталонное значение реквизита
  // в синхронизируемый реквизит
  oleRef_FZ_2.УстановитьАтрибут(''+idAttr,Trim(oleVal));
end;
```

Для перечислений (используя методы `ЗначениеВСтроку`, `ЗначениеИзСтроки` и `EvalExpr`):

```
if (AnsiUpperCase(sType1)=AnsiUpperCase('Перечисление')) then begin
  // Это — перечисление.
  // Прочитаем идентификатор значения реквизита
  idVal:=Trim(oleVal.Идентификатор);
  // Формируем строку для выполнения с помощью метода EvalExpr()
  Sx:='Перечисление.ПолучитьАтрибут("'+sKind1+'")';
  Sx:=Sx+'.ЗначениеПоИдентификатору("'+idVal+'")';
  Sx:='ЗначениеИзСтроки(ЗначениеВСтроку('+Sx+'))';
  // Получаем ссылку на значение перечисления
  // в синхронизируемой базе данных
  oleVal:=ole1C_2.EvalExpr(Sx);
  // Записываем новое значение
  // в синхронизируемый реквизит
  oleRef_FZ_2.УстановитьАтрибут(''+idAttr,oleVal);
end;
```

Для реквизитов типа `Справочник` (учитывая, что справочник, элементы которого являются множеством значений для данного реквизита, может иметь владельца):

```
if (AnsiUpperCase(sType1)=AnsiUpperCase('Справочник')) then begin
  // Это — Справочник.
  // Прочитаем код элемента справочника (значение реквизита)
  idVal:=Trim(oleVal.Код);
  // Определим владельца элемента справочника (значение реквизита)
  idOwner:='';
  if (oleVal.Владелец.Выбран*1)>0 then begin
     // Читаем код владельца
     idOwner:=Trim(oleVal.Владелец.Код);
  end;
  // Если у этого реквизита есть владелец,
  // позиционируемся на нем
```

```
  oleOwner2:=UnAssigned; // Справочник-владелец
  oleOwnerTekEl2:=UnAssigned; // "ТекущийЭлемент" владельца
  if length(idOwner)>0 then begin
  // Владелец существует
    oleOwner2:=
      ole1C_2.CreateObject('Справочник.'+Trim(oleVal.Владелец.Вид));
    if oleOwner2.НайтиПоКоду(''+idOwner,0) then begin
    // Считываем ссылку на элемент-владелец
      oleOwnerTekEl2:=oleOwner2.ТекущийЭлемент;
    end;
  end;
  // Ищем текущий элемент справочника с кодом idVal
  // в синхронизируемой базе данных
  oleRef2:=ole1C_2.CreateObject('Справочник.'+Trim(oleVal.Вид));
  if not VarIsEmpty(oleOwnerTekEl2) then begin
     // Если есть владелец — учитываем его
     oleRef2.ИспользоватьВладельца(oleOwnerTekEl2,0);
  end;
  // Ищем новое значение реквизита
  if (oleRef2.НайтиПоКоду(''+idVal,1)*1)>0 then begin
  // Записываем эталонное значение реквизита
  // в синхронизируемый реквизит
    oleRef_FZ_2.УстановитьАтрибут(''+idAttr,oleRef2.ТекущийЭлемент);
  end;
end;
```

***Примечание***

Естественно, что в случае реквизитов других типов (например, `Документ`) этот код должен быть функционально расширен.

Если синхронизируемый реквизит периодический — то используем объект `Периодический` (как мы делали это в *разд. 4.7.1*). Сначала в синхронизируемом справочнике удаляем все значения реквизита (которые не связаны с документами):

```
// Создаем объект "Периодический" в синхронизируемой базе данных
olePeriod2:= ole1C_2.CreateObject('Периодический');
// Устанавливаем фильтр по синхронизируемому реквизиту
```

```
// и элементу справочника
if (olePeriod2.ИспользоватьОбъект(''+idAttr,
           oleRef_FZ_2.ТекущийЭлемент)*1) > 0 then begin
  // Удаляем ВСЕ значения данного реквизита,
  // кроме тех, что установлены документами
  // при проведении
  if (olePeriod2.ВыбратьЗначения*1) >0 then begin
    while (olePeriod2.ПолучитьЗначение*1) >0 do
      begin
      // Пропускаем значения, созданные документами
      if (olePeriod2.ТекущийДокумент.Выбран*1)<=0 then begin
        if (olePeriod2.Удалить*1)>0 then begin
        // Удачно
        end;
      end;
    end;
  end;
end;
```

Затем копируем все значения из периодического реквизита элемента-эталона в синхронизируемый элемент *без ссылки на документы*:

```
// Создаем объект "Периодический" в эталонной базе данных
olePeriod1:= ole1C_1.CreateObject('Периодический');
if (oleperiod1.ИспользоватьОбъект(''+idAttr,
    oleRef_FZ_1.ТекущийЭлемент)*1) > 0 then begin
  // Копируем все значения периодического реквизита-эталона
  if (olePeriod1.ВыбратьЗначения*1) >0 then begin
    while (olePeriod1.ПолучитьЗначение*1) >0 do
      begin
      // Проверяем корректность даты в значении реквизита-эталона
        if sp_IsDate(Trim(olePeriod1.ДатаЗнач)) then begin
        // Сразу же устанавливаем дату значения
          olePeriod2.ДатаЗнач:=Trim(olePeriod1.ДатаЗнач);
          // Для упрощения — не будем отслеживать
          // свойство "ТекущийДокумент".
          // Читаем значение реквизита-эталона
          oleVal:=olePeriod1.Значение;
           ...
```

```
// Далее, почти так же, как и ранее (для непериодических реквизитов).
// В соответствии с типом значения...
          ...
          // Присваиваем значение синхронизируемому реквизиту
          olePeriod2.Значение:= <Здесь соответствующее значение>;
          // Записываем изменение в синхронизируемую базу данных
          olePeriod2.Записать;
```

Вот, собственно, и все. В принципе — ничего "военного". Все это было рассмотрено в предыдущих разделах. Но потребуются существенные усилия и точность в программировании.

***Примечание***

Напомню, что представленные примеры исходных текстов — это упрощенная иллюстрация к рассматриваемым вопросам. Назначение этих примеров — дать представление о возможном подходе при решении задач информационного обмена с системой 1С:Предприятие. При работе с реальными базами 1С может потребоваться их дальнейшая доработка.

# Глава 5

# Расширение функционала 1С

Как отмечалось выше, система 1С:Предприятие обладает собственными (штатными) средствами, позволяющими произвести конфигурирование информационной структуры и настройку функционала для решения прикладных задач с учетом особенностей конкретной организации. Тем не менее, для решения специальных задач предусмотрена возможность расширения функциональности системы за счет использования *технологии создания внешних компонент*. Термин "внешняя компонента" адекватен, по сути, термину "сервер автоматизации" или "сервер OLE Automation" *(см. главу 1)*.

Декларированные фирмой 1С возможности внешних компонент:

- расширение встроенного языка системы (новые объекты агрегатного типа);
- добавление страницы свойств в параметры системы 1С:Предприятие;
- сохранение параметров внешней компоненты (с использованием механизмов системы 1С:Предприятие);
- создание дополнительных окон в окне системы 1С:Предприятие;
- доступ к функциям системы 1С:Предприятие (с помощью механизма OLE Automation);
- вызов процедур обработки событий (в контексте системы 1С:Предприятие), генерируемых внешней компонентой;
- доступ к строке состояния программы 1С:Предприятие.

Основное назначение внешних компонент: сопряжение с различным торговым оборудованием, нестандартное отображение данных, дополнительные (специфические) функции и т. д.

Есть две версии внешних компонент:

- версия 1.0 — совместима с системой 1С:Предприятие версий 7.5 и 7.7;
- версия 2.0 — совместима с системой 1С:Предприятие версии 7.7.

Все создаваемые внешние компоненты должны следовать единому стандарту. Упростить процедуру создания стандартных внешних компонент позволяют специальные шаблоны, разработанные фирмой 1С для разных языков программирования (Borland Delphi, Microsoft Visual C++, Microsoft Visual Basic).

***Примечание***

Более подробную информацию о технологии создания внешних компонент можно найти в специальной литературе по системе 1С:Предприятие.

В этой главе мы рассмотрим процесс создания внешних компонент (с использованием шаблонов фирмы 1С) на конкретных примерах.

# 5.1. Общие сведения

Как отмечалось в *главе 1*, COM (Component Object Model — модель составных объектов) — одна из базовых технологий Windows. Основная цель COM — возможность экспорта объектов, создаваемых внутри приложения. То есть одно приложение создает объект, а другие приложения используют свойства и методы (сервисы) экспортируемого объекта. При этом не имеет принципиального значения, на каком именно языке программирования разрабатывались "взаимодействующие" приложения. Главное, чтобы был соблюден стандарт COM.

Приложение, поддерживающее технологию COM, должно иметь в своем составе хотя бы один *COM-объект*. Объект является *экземляром какого-то класса* (переменная объектного типа). Существенное отличие COM-объекта от "обычного" объекта в том, что COM-объект должен иметь хотя бы один интерфейс (их количество концептуально не органичено). У COM-объектов есть также отличия в объектном механизме наследования. COM-объекты поддерживают наследование интерфейсов.

*Интерфейс* (interface) — это средство согласованного взаимодействия экспортируемого COM-объекта и приложения-клиента, которое обращается к свойствам и методам экспортируемого COM-объекта. Интерфейс имеет два идентифицирующих его атрибута — имя интерфейса и его GUID *(см. главу 1)*. Естественно, что GUID для каждого интерфейса должен быть уникальным. В среде Windows GUID формируется функцией `CoCreateGuid`. Среда Delphi позволяет формировать GUID прямо в тексте исходного кода модуля. Для этого достаточно нажать сочетание клавиш: <Ctrl>+<Shift>+<G>.

Любой COM-объект должен иметь базовый интерфейс `IUnknown`. С помощью этого интерфейса можно получить список всех интерфейсов данного COM-объекта. Интерфейс `IUnknown` наследуется всеми остальными интерфейсами.

У интерфейса `IUnknown` есть 3 метода:

- `QueryInterface` — возвращает указатель на требуемый интерфейс;
- `_AddRef` — поддерживает механизм учета ссылок (увеличивает счетчик ссылок на единицу);
- `_Release` — поддерживает механизм учета ссылок (уменьшает счетчик ссылок на единицу, как только счетчик ссылок обнуляется — COM-объект "самоуничтожается").

Для создания и запуска COM-объекта (как экземпляра класса) используется *фабрика класса*. Для каждого класса должна быть своя фабрика класса.

*COM-сервер* — это исполняемый модуль (DLL-файл или приложение), имеющий в своем составе COM-объект(ы). COM-сервер должен быть зарегистрирован в операционной системе специальным образом (в системном реестре Windows). Есть разные способы регистрации COM-серверов. В частности, в состав Windows входит специальная утилита для регистрации COM-серверов — RegSvr32.EXE.

Как говорилось в *главе 1*, есть 3 вида серверов:

- внутрипроцессный (in-process) сервер — в виде DLL-библиотеки;
- внепроцессный (out of process) сервер — в виде отдельного приложения;
- удаленный (remote) сервер — запущен на другом (относительно клиента) компьютере (очевидно, что внутрипроцессный сервер не может служить удаленным сервером).

На COM-технологии базируется технология OLE Automation. При этом:

- COM-серверы должны поддерживать интерфейс `IDispatch` — важнейший элемент технологии OLE Automation (поддержка механизма позднего связывания):

  ```
  IDispatch = interface(IUnknown)
  ```

- COM-серверы должны поддерживать типы данных, используемые в технологии OLE Automation (например `byte`, `variant`, `date`, `bstr`, `hresult`, `long` и др.);
- все методы интерфейсов должны быть процедурами, поддерживающими соглашение о вызове safecall, или функциями, поддерживающими соглашение о вызове safecall и возвращающими значение типа `HResult`.

При управлении сервером автоматизации используются *раннее связывание* (все обращения к методам интерфейсов проверяются на корректность на этапе компиляции программы-клиента) и *позднее связывание* (все обращения к методам интерфейсов проверяются на корректность на этапе выполнения программы-клиента).

Информация об интерфейсах, а также их свойствах и методах, содержится в библиотеке типов (файлы *.TLB). При создании библиотеки типов используется специальный язык IDL (Interface Definition Language).

***Примечание***

В общем случае наличие библиотеки типов не обязательно. Но для реализации механизма раннего связывания она необходима.

***Примечание***

COM-технология — очень объемная, важная и сложная тема, охватывающая различные аспекты создания и взаимодействия приложений в среде Windows, и в рамках данной книги она детально не рассматривается. Подробную информацию о технологии COM можно найти в специальной литературе.

## 5.2. Внутрипроцессный сервер автоматизации

Внутрипроцессный сервер автоматизации реализуется в виде специального DLL-модуля (библиотеки). В среде Delphi такой модуль создается следующим образом.

Нужно выбрать пункт **File > New**, в открывшемся диалоговом окне перейти на вкладку **ActiveX** и выбрать значок **ActiveX Library** (рис. 5.1).

После этого Delphi сгенерирует соответствующий код (листинг 5.1).

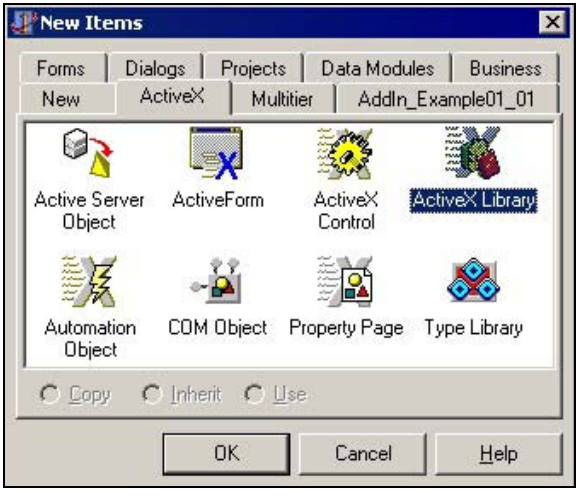



**Рис. 5.1.** Выбор типа DLL-библиотеки для создания сервера автоматизации

**Листинг 5.1. Код, созданный Delphi при выборе значка *ActiveX Library***

```
library Project1;
uses
  ComServ;
exports
  DllGetClassObject,
  DllCanUnloadNow,
  DllRegisterServer,
  DllUnregisterServer;

{$R *.RES}

begin
end.
```

В табл. 5.1 представлена информация о назначении функций из листинга 5.1. Реализация этих функций содержится в файле ComServ.PAS.

***Таблица 5.1.*** *Назначение функций из листинга 5.1*

| Функция | Назначение |
|---|---|
| DllGetClassObject | Создание фабрики классов и предоставление доступа к библиотеке классов |
| DllCanUnloadNow | Проверка возможности выгрузки DLL-библиотеки из памяти |
| DllRegisterServer | Регистрация сервера в системном реестре Windows |
| DllUnregisterServer | Удаление информации о сервере из реестра Windows |

Для создания объекта нужно вызвать диалоговое окно (см. рис. 5.1) и выбрать значок **Automation Object**. На экране появится диалоговое окно для ввода имени класса COM-объекта (рис. 5.2).

Поле **CoClass Name** содержит имя класса COM-объекта (это имя будет зарегистрировано в системном реестре Windows).

Раскрывающийся список **Instancing** позволяет выбрать одно из следующих значений:

- **Internal** — фабрика классов не будет регистрироваться в системном реестре, следовательно, этот класс не будет доступен для клиентских приложений;

- **Single Instance** — для каждого клиента будет запускаться экземпляр COM-сервера;
- **Multiple Instance** — запускается один экземпляр COM-сервера для всех клиентов.

**Рис. 5.2.** Ввод имени класса COM-объекта

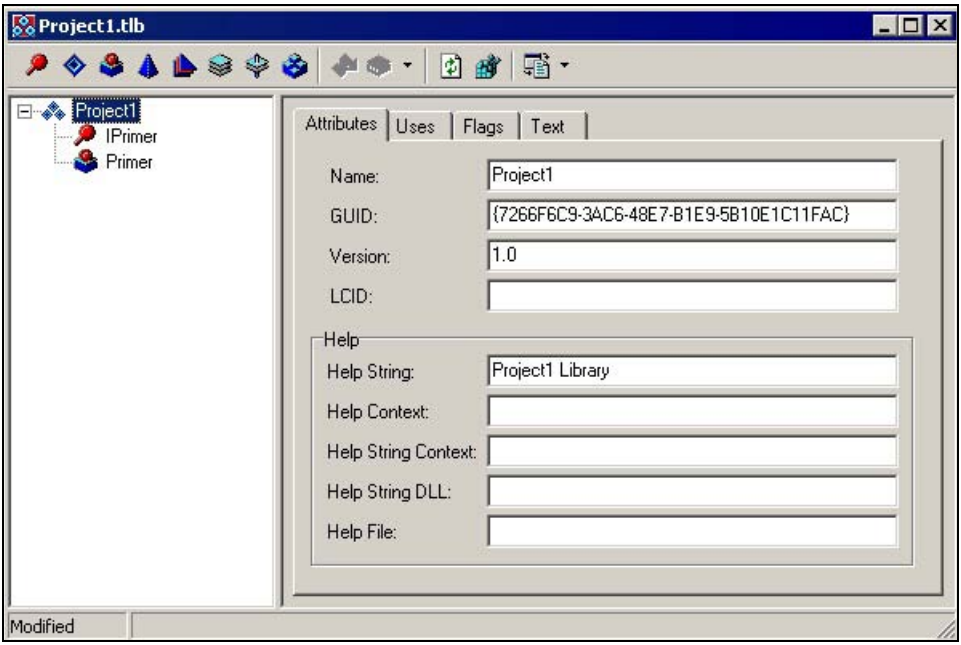



**Рис. 5.3.** Редактор библиотеки типов

Раскрывающийся список **Threading Model** позволяет выбрать одно из следующих значений:

- **Single** — однопоточная модель;
- **Apartment** — модель разделенных потоков;
- **Free** — модель свободных потоков;
- **Both** — модель смешанных потоков.

Флажок **Generate Event support code** служит для поддержки механизма нотификационных сообщений.

После нажатия кнопки **OK** (см. рис. 5.2) на экране появится диалоговое окно редактирования библиотеки типов создаваемого сервера (рис. 5.3).

В диалоговом окне редактирования библиотеки типов создаются свойства и методы COM-объекта, которые будут доступны клиентским приложениям (контроллерам автоматизации) (рис. 5.4).

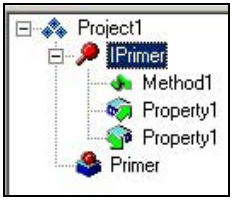



**Рис. 5.4.** Новый метод и новое свойство COM-объекта сервера автоматизации

Среда Delphi автоматически сгенерирует соответствующий код (листинг 5.2).

**Листинг 5.2. Код, созданный Delphi при выборе значка *Automation Object***

```
unit Unit1;
interface
uses
 ComObj, ActiveX, Project1_TLB, StdVcl;
type
 TPrimer = class(TAutoObject, IPrimer)
 protected
  procedure Method1; safecall;
  function Get_Property1: Integer; safecall;
  procedure Set_Property1(Value: Integer); safecall;
  { Protected declarations }
```

```
 end;

implementation
uses ComServ;

procedure TPrimer.Method1;
begin

end;

function TPrimer.Get_Property1: Integer;
begin

end;

procedure TPrimer.Set_Property1(Value: Integer);
begin

end;

initialization
 TAutoObjectFactory.Create(ComServer, TPrimer, Class_Primer,
  ciMultiInstance, tmApartment);
end.
```

Как видите, для написания кода реализации заданных свойств и методов COM-объекта оставлены соответствующие пустые "заглушки".

Здесь следует отметить одну существенную деталь. Свойство `ProgID` COM-объекта в данном случае будет автоматически сформировано следующим образом:

```
Project1.Primer
```

То есть в первой части (до точки) используется имя DLL-модуля (в данном случае), а во второй части — имя COM-объекта.

> ***Примечание***
>
> Далее мы увидим, что при создании внешней компоненты для системы 1С:Предприятие в первой части свойства `ProgID` всегда должна быть цепочка символов `AddIn`. Но это не означает, что DLL-модуль обязательно должен называться *AddIn.DLL*.

Чтобы добавить страницу свойств сервера, нужно вызвать диалоговое окно (см. рис. 5.1) и выбрать значок **Property Page**. Delphi создаст форму `TPropertyPage`:

```
TPropertyPage = class(TCustomForm)
```

и сгенерирует соответствующий код (листинг 5.3).

**Листинг 5.3. Код, сгенерированный Delphi при создании формы `TPropertyPage`**

```
unit Unit1;
interface
uses SysUtils, Windows, Messages, Classes, Graphics, Controls, StdCtrls,
  ExtCtrls, Forms, ComServ, ComObj, StdVcl, AxCtrls;

type
  TPropertyPage1 = class(TPropertyPage)
  private
    { Private declarations }
  protected
    { Protected declarations }
  public
    { Public declarations }
    procedure UpdatePropertyPage; override;
    procedure UpdateObject; override;
  end;

const
  Class_PropertyPage1: TGUID = '{F8E326B1-5FB9-4973-B685-A5E084D530B7}';

implementation

{$R *.DFM}

procedure TPropertyPage1.UpdatePropertyPage;
begin
  { Update your controls from OleObject }
end;

procedure TPropertyPage1.UpdateObject;
```

```
begin
  { Update OleObject from your controls }
end;

initialization
  TActiveXPropertyPageFactory.Create(
    ComServer,
    TPropertyPage1,
    Class_PropertyPage1);
end.
```

Как видите, процесс создания внутрипроцессного сервера автоматизации в среде Delphi максимально упрощен и не представляет трудностей.

> ***Примечание***
>
> Более подробную информацию о создании внутрипроцессных серверов автоматизации можно найти в специальной литературе.

В *главе 1* говорилось, система 1С поддерживает механизм OLE Automation как сервер и как клиент. По отношению к внешней компоненте программа 1С выполняет роль клиента (контроллера автоматизации). Следовательно, здесь применимы все приемы и способы программирования серверов автоматизации, разработанные для использования этой технологии в других приложениях. Тем не менее, при создании внешних компонент для системы 1С:Предприятие есть некоторые нюансы.

Рассмотрим процесс создания внешних компонент для программы 1С:Предприятие на конкретных примерах, используя шаблоны, разработанные фирмой 1С для Delphi.

Прежде всего следует отметить, что все интерфейсы системы 1С:Предприятие определены в файле AddInLib.PAS. То есть этот модуль должен присутствовать в директиве `uses` других модулей, использующих эти интерфейсы.

Создадим внешнюю компоненту версии 1.0 с использованием соответствующего шаблона (sample).

> ***Примечание***
>
> Описание шаблона приведено в руководстве 1С по технологии создания внешних компонент.

Прежде чем приступить к созданию внешней компоненты, нужно придумать имя компоненты (вторая часть имени в свойстве `ProgID`) и имя DLL-файла

компоненты (предполагается, что назначение компоненты, т. е. ее функциональность, уже определена). В нашем случае имя компоненты — `AddIn_Example01`, а имя DLL-файла — AddIn_Example01_01.DLL.

Создадим новый каталог (например, AddIn_Example01_01) и скопируем туда все файлы шаблона. Переименуем файл проекта шаблона AddIn.DPR в AddIn_Example01_01.DPR.

> ***Примечание***
>
> Текст модуля AddIn.DPR шаблона практически идентичен тексту, представленному в листинге 5.1.

Остальные файлы проекта (AddInLib.PAS, AddInObj.PAS, PropPage.*) можно оставить без изменения.

> ***Примечание***
>
> Исходные тексты этого примера приведены на сопроводительном компакт-диске книги в папке \Part_1\Chapter_05\Topic_5_2\AddIn_Example01_01.PAS.
>
> Здесь файлы AddInObj.PAS и PropPage.* переименованы в AddInObj_01.PAS и PropPage_01.* соответственно.

Откроем файл AddInObj_01.PAS.

Как видите, OLE-объект (`TAddInObject`) здесь унаследован не от `TAutoObject` (листинг 5.2), а от `TComObject`. Мотивация разработчика шаблона (фирма 1С): если `TAddInObject` наследуется от `TAutoObject`, это приведет к созданию и использованию библиотеки типов, что усложнит шаблон.

В табл. 5.2 перечислены основные методы OLE-объекта `TAddInObject`.

***Таблица 5.2.*** *Назначение методов внешней компоненты (шаблон Sample)*

| Метод | Назначение |
|---|---|
| `Init` | Инициализирует объект компоненты. При успешном выполнении возвращает значение `S_OK`, в противном случае — значение `E_FAIL` |
| `Done` | Вызывается при завершении работы с объектом компоненты. При успешном выполнении возвращает значение `S_OK` |
| `GetInfo` | Вызывается программой 1С:Предприятие для получения информации о компоненте (в частности, о версии компоненты). При успешном выполнении возвращает значение `S_OK` |
| `RegisterExtensionAs` | Определяет вторую часть свойства `ProgID` (т. е. имя компоненты). При успешном выполнении возвращает значение `S_OK` |

***Таблица 5.2*** *(продолжение)*

| Метод | Назначение |
|---|---|
| GetNProps | Определяет количество свойств компоненты. При успешном выполнении возвращает значение S_OK |
| FindProp | Определяет порядковый номер свойства по его имени. При успешном выполнении возвращает значение S_OK, в противном случае — значение S_FALSE |
| GetPropName | Определяет имя свойства по его номеру. При успешном выполнении возвращает значение S_OK, в противном случае — значение S_FALSE |
| GetPropVal | Определяет значение свойства по его номеру. При успешном выполнении возвращает значение S_OK, в противном случае — значение S_FALSE |
| SetPropVal | Присваивает свойству новое значение. При успешном выполнении возвращает значение S_OK, в противном случае — значение S_FALSE |
| IsPropReadable | Определяет, доступно ли свойство для чтения. При успешном выполнении возвращает значение S_OK, в противном случае — значение S_FALSE |
| IsPropWritable | Определяет, доступно ли свойство для записи. При успешном выполнении возвращает значение S_OK, в противном случае — значение S_FALSE |
| GetNMethods | Определяет количество методов компоненты. При успешном выполнении возвращает значение S_OK, в противном случае — значение S_FALSE |
| FindMethod | Определяет порядковый номер метода по его имени. При успешном выполнении возвращает значение S_OK, в противном случае — значение S_FALSE |
| GetMethodName | Определяет имя метода по его номеру. При успешном выполнении возвращает значение S_OK, в противном случае — значение S_FALSE |
| GetNParams | Определяет количество параметров заданного метода. При успешном выполнении возвращает значение S_OK, в противном случае — значение S_FALSE |
| GetParamDefValue | Определяет значение по умолчанию для заданного параметра заданного метода. При успешном выполнении возвращает значение S_OK, в противном случае — значение S_FALSE |
| HasRetVal | Определяет, является ли заданный метод функцией (возвращает значение) или процедурой (не возвращает значения). При успешном выполнении возвращает значение S_OK, в противном случае — значение S_FALSE |

***Таблица 5.2*** *(окончание)*

| Метод | Назначение |
|---|---|
| CallAsProc | Реализует вызов заданного метода как процедуры. При успешном выполнении возвращает значение S_OK, в противном случае — значение S_FALSE |
| CallAsFunc | Реализует вызов заданного метода как функции. При успешном выполнении возвращает значение S_OK, в противном случае — значение S_FALSE |
| GetPages | Используется для привязки к странице свойств (TAddInPropPage). При успешном выполнении возвращает значение S_OK |
| GetIDsOfNames | Позволяет получить номер метода объекта (программы 1С:Предприятие как сервера OLE Automation) по имени метода (используется перед вызовом метода Invoke интерфейса IDispatch). Если не реализовано, возвращает значение E_NOTIMPL |
| Invoke | Осуществляет вызов метода объекта (программы 1С:Предприятие как сервера OLE Automation) по номеру. Если не реализовано, возвращает значение E_NOTIMPL |

***Примечание***

Более подробную информацию о шаблонах (для разработки внешних компонент) фирмы 1С можно получить из специальной литературы по системе 1С:Предприятие.

Прежде всего, создадим новый GUID для новой компоненты и присвоим его значение переменной CLSID_AddInObject (листинг 5.4).

**Листинг 5.4. Новый GUID для новой внешней компоненты**

```
const
   CLSID_AddInObject : TGUID = '{AE5D9B58-F9E3-4A38-9262-621D1CEAAC97}';
```

Затем определим три константы (листинг 5.5), которые избавят нас от необходимости постоянно искать и менять определенные строки в тексте модуля при переходе от одной версии примера к другой.

**Листинг 5.5. Константы, определяющие имя внешней компоненты и имя DLL-файла компоненты**

```
Const
  // Имя компоненты (вторая часть свойства ProgID)
  ExtensionName_Comp = 'AddIn_Example01';
```

```
  // Суффикс для имени DLL-файла версии примера
  ExtensionName_Comp_Vers = '_01';
  // ИМЯ DLL-файла внешней компоненты
  dllName_Comp = ExtensionName_Comp+ExtensionName_Comp_Vers;
```

В тексте модуля AddInObj_01.PAS сделаем соотвтетствующие замены (листинг 5.6).

**Листинг 5.6. Замены в тексте модуля AddInObj_01.PAS**

```
function TAddInObject.Init(pConnection: IDispatch): HResult; stdcall;
  ...
  if (iRes = S_OK) then begin
     //pProfile.RegisterProfileAs('Sample AddIn Profile Name');
     pProfile.RegisterProfileAs(ExtensionName_Comp+' Profile Name');
  ...
end;
  ...
function TAddInObject.RegisterExtensionAs(var bstrExtensionName:
  WideString): HResult; stdcall;
begin
  //bstrExtensionName := 'AddInExtension';
  bstrExtensionName := ExtensionName_Comp;
  RegisterExtensionAs := S_OK;
end;
  ...
initialization
ComServer.SetServerName('AddIn');
TComObjectFactory.Create(ComServer,TAddInObject,CLSID_AddInObject,
  //'AddIn','V7 AddIn 2.0',ciMultiInstance);
  dllName_Comp,'V7 AddIn 2.0',ciMultiInstance);
end.
```

Здесь следует отметить следующую важную деталь. Версия компоненты должна быть определена в методе `GetInfo` следующим образом (листинг 5.7).

**Листинг 5.7. Определение версии внешней компоненты**

```
function TAddInObject.GetInfo(var pInfo: PSafeArray{(OleVariant)}):
                                                    HResult; stdcall;
var
```

```
  varInfo : OleVariant;
begin
  varInfo := '2000';
  PutNParam(pInfo,0,varInfo);
  GetInfo := S_OK;
end;
```

Для внешней компоненты версии 1.00 в переменную `varInfo` должна быть записана строка `'1000'`, а версии 2.00 — `'2000'`. Поскольку все примеры данной книги ориентированы на 1С:Предприятие версии 7.7, которая поддерживает внешние компоненты обеих версий (1.00 и 2.00), то нет необходимости менять значение этой переменной.

В принципе, на этом подготовительные операции закончены, и можно перейти к программированию функциональности компоненты. Чтобы еще более упростить первый пример, пришлось удалить из данного модуля шаблона практически все, кроме реализации свойства `Enabled` (`Включен`) внешней компоненты. В дальнейшем мы будем постепенно наращивать функциональность.

Функциональность первой версии примера использования шаблона (sample) будет заключаться только в том, что внешняя компонента будет иметь свойство `Включен` и два метода — `Включить` и `Выключить`.

Свойство `Включен` определено на странице свойств (форма `TAddInPropPage`) в виде Delphi-компонента **Компонента включена** (рис. 5.5) следующим образом (листинг 5.8).

**Листинг 5.8. Определение свойства внешней компоненты**

```
type
  TAddInPropPage = class(TPropertyPage)
    IsEnabled: TCheckBox; // "Компонента включена"
    ...
  end;
```

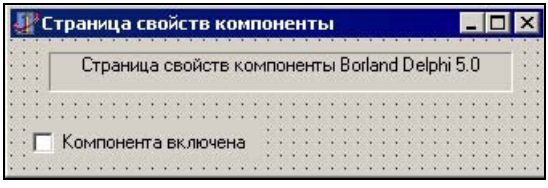



**Рис. 5.5.** Определение нового свойства компоненты на странице свойств

Следующий шаг — объявить *имена* свойств и методов внешней компоненты так, чтобы их можно было использовать в контексте встроенного языка системы 1С:Предприятие. В рамках рассматриваемого шаблона это делается следующим образом (листинг 5.9).

**Листинг 5.9. Определение имен свойств и методов внешней компоненты для программы 1С:Предприятие**

```
resourcestring
  strIsEnabled = 'IsEnabled,Включен'; // Свойство
  strEnable = 'Enable,Включить';      // Метод
  strDisable = 'Disable,Выключить';   // Метод
```

Как видите, англоязычное и русскоязычное написания имен методов (свойств) разделены запятой.

Далее нужно перечислить свойства и методы так, как это указано в листинге 5.10.

**Листинг 5.10. Свойства и методы внешней компоненты**

```
Type
  TProperties = ( propIsEnabled, LastProp );          // Свойства
  TMethods = ( methEnable, methDisable, LastMethod ); // Методы
```

Здесь важно отметить, что свойство `LastProp` и метод `LastMethod` должны быть всегда последними в списке перечислений (`TProperties` и `TMethods`). Это связано с тем, что они участвуют в определении количества свойств (`GetNProps`) и методов (`GetNMethods`) внешней компоненты следующим образом (листинг 5.11 и 5.12) .

**Листинг 5.11. Определение количества свойств внешней компоненты**

```
function TAddInObject.GetNProps(var plProps: Integer): HResult; stdcall;
begin
  plProps := Integer(LastProp);
  GetNProps := S_OK;
end;
```

**Листинг 5.12. Определение количества методов внешней компоненты**

```
function TAddInObject.GetNMethods(var plMethods: Integer): HResult; stdcall;
begin
  plMethods := Integer(LastMethod);
  GetNMethods := S_OK;
end;
```

Реализация загрузки параметров OLE-объекта внешней компоненты приведена в листинге 5.13.

**Листинг 5.13. Загрузка параметров OLE-объекта внешней компоненты**

```
function TAddInObject.LoadProperties: Boolean;
var
 iRes : Integer;
 varRead : OleVariant;
begin
 VarClear(varRead);
 { SImply cast OleVariant to Integer (VT_I4) }
 varRead := 0;
 iRes := pProfile.Read('Enabled:0',varRead,nil);
 if (iRes <> S_OK) then begin
    LoadProperties := False;
    Exit;
 end;
 boolIsEnabled := varRead;
 LoadProperties := True;
end;
```

Реализация сохранения параметров OLE-объекта внешней компоненты приведена в листинге 5.14.

**Листинг 5.14. Сохранение параметров OLE-объекта внешней компоненты**

```
procedure TAddInObject.SaveProperties;
var
  varSave : OleVariant;
begin
```

```
  varSave := boolIsEnabled;
  pProfile.Write('Enabled:0',varSave);
end;
```

Определение порядкового номера свойства по его имени приведено в листинге 5.15.

**Листинг 5.15. Определение порядкового номера свойства по его имени**

```
function TAddInObject.FindProp(const bstrPropName: WideString; var
plPropNum: Integer): HResult; stdcall;
begin
  plPropNum := -1;
  //----------------------------
  // Соответствие имени свойства и его порядкового номера
  if (bstrPropName = TermString(strIsEnabled,0)) then plPropNum := 0;
  if (bstrPropName = TermString(strIsEnabled,1)) then plPropNum := 0;
  //----------------------------
  if (plPropNum = -1) then begin
     FindProp := S_FALSE;
     Exit;
  end;
  FindProp := S_OK;
end;
```

Здесь функция `TermString` выделяет из первого параметра (типа `String`) англоязычное или русскоязычное написание свойства (см. листинг 5.9) в зависимости от значения второго параметра. В обоих случаях метод `FindProp` определяет одно и то же значение для переменной `plPropNum`. Это значение (в данном случае ноль) должно строго соответствовать порядку следования значений (в данном случае `propIsEnabled`) перечисления `TProperties` (листинг 5.10). В данном случае свойство:

```
'IsEnabled,Включен';
```

установлено в соответствие значению перечисления `propIsEnabled`:

```
( propIsEnabled, LastProp );
```

Такому же правилу нужно следовать при написании кода метода `FindMethod` (листинг 5.16).

**Листинг 5.16. Определение порядкового номера метода по его имени**

```
function TAddInObject.FindMethod(const bstrMethodName: WideString;
    var plMethodNum: Integer): HResult; stdcall;
begin
  plMethodNum := -1;
  //----------------------------
  // Соответствие имени метода и его порядкового номера
  if (bstrMethodName = TermString(strEnable,0)) then plMethodNum := 0;
  if (bstrMethodName = TermString(strEnable,1)) then plMethodNum := 0;
  if (bstrMethodName = TermString(strDisable,0)) then plMethodNum := 1;
  if (bstrMethodName = TermString(strDisable,1)) then plMethodNum := 1;
  //----------------------------
  if (plMethodNum = -1) then begin
     FindMethod := S_FALSE;
     Exit;
  end;
  FindMethod := S_OK;
end;
```

Здесь имя метода внешней компоненты `Включить` (так, как оно будет использоваться в контексте встроенного языка программирования системы 1С:Предприятие) соответствует значению `methEnable` перечисления `TMethods`. А имя метода `Выключить` соответствует значению `methDisable` (см. листинг 5.9 и 5.10).

Определение имени свойства по его порядковому номеру приведено в листинге 5.17.

**Листинг 5.17. Определение имени свойства по его номеру**

```
function TAddInObject.GetPropName(lPropNum, lPropAlias: Integer;
     var pbstrPropName: WideString): HResult; stdcall;
begin
  pbstrPropName := '';
  case TProperties(lPropNum) of
    propIsEnabled: // Значение перечисления TProperties
      begin
        pbstrPropName := TermString(strIsEnabled,lPropAlias);
      end;
```

```
    else
      begin
        GetPropName := S_FALSE;
        Exit;
      end;
  end;
  GetPropName := S_OK;
end;
```

Здесь комментарии излишни.

По такому же принципу реализован метод `GetMethodName` (листинг 5.18).

**Листинг 5.18. Определение имени метода по его номеру**

```
function TAddInObject.GetMethodName(lMethodNum, lMethodAlias: Integer;
     var pbstrMethodName: WideString): HResult; stdcall;
begin
  pbstrMethodName := '';
  case TMethods(lMethodNum) of
    methEnable: // Значение перечисления TMethods
      begin
        pbstrMethodName := TermString(strEnable,lMethodAlias);
      end;
    methDisable: // Значение перечисления TMethods
      begin
        pbstrMethodName := TermString(strDisable,lMethodAlias);
      end;
    else
      begin
        GetMethodName := S_FALSE;
        Exit;
      end;
  end;
  GetMethodName := S_OK;
end;
```

Реализация определения значения свойства по его номеру представлена в листинге 5.19.

**Листинг 5.19. Определение значения свойства по его номеру**

```
function TAddInObject.GetPropVal(lPropNum: Integer;
     var pvarPropVal: OleVariant): HResult; stdcall;
begin
  VarClear(pvarPropVal);
  case TProperties(lPropNum) of
    propIsEnabled: // Значение перечисления TProperties
      begin
        pvarPropVal := boolIsEnabled;
      end;
    else
      begin
        GetPropVal := S_FALSE;
        Exit;
      end;
  end;
  GetPropVal := S_OK;
end;
```

Здесь следует сделать некоторые пояснения.

В нашем случае свойство `Включен` (`IsEnabled`) на странице свойств `TAddInPropPage` определено как тип `Boolean` (свойство `Checked` Delphi-компонента `IsEnabled : TCheckBox`). Но в контексте встроенного языка системы 1С:Предприятие нет такого типа данных *(см. главу 4)*. Поэтому у объекта `TAddInObject` объявлено свойство типа `Integer`:

```
boolIsEnabled : Integer;
```

значение которого и считывается из программы 1С:Предприятие. То же самое относится к реализации метода `SetPropVal` (листинг 5.20).

**Листинг 5.20. Присвоение значения свойству по его номеру**

```
function TAddInObject.SetPropVal(lPropNum: Integer;
     var varPropVal: OleVariant): HResult; stdcall;
begin
  case TProperties(lPropNum) of
    propIsEnabled: // Значение перечисления TProperties
      begin
        boolIsEnabled := varPropVal;
```

```
      end;
    else
      begin
        SetPropVal := S_FALSE;
        Exit;
      end;
  end;
  SetPropVal := S_OK;
end;
```

Здесь следует отметить, что для связи со страницей свойств и обмена данными используется интерфейс `IPropertyLink` (листинг 5.21).

**Листинг 5.21. Объявление интерфейса `IpropertyLink` (модуль AddInLib.PAS)**

```
IPropertyLink = interface(IUnknown)
  ['{B59789E1-2E8B-11d1-A4E7-004095E1DAEA}']
  function get_Enabled(var boolIsEnabled : Integer): HResult; stdcall;
  function put_Enabled(boolIsEnabled : Integer): HResult; stdcall;
end;
```

Реализация его методов представлена в листинге 5.22.

**Листинг 5.22. Реализация методов интерфейса `IPropertyLink`**

```
function TAddInObject.get_Enabled(var IsEnabled : Integer):
                                                  HResult; stdcall;
begin
  IsEnabled := boolIsEnabled;
  get_Enabled := S_OK;
end;

function TAddInObject.put_Enabled(IsEnabled : Integer): HResult; stdcall;
begin
  boolIsEnabled := IsEnabled;
  put_Enabled := S_OK;
end;
```

Реализация связи со страницей свойств представлена в листинге 5.23.

**Листинг 5.23. Связь со страницей свойств**

```
procedure TAddInPropPage.UpdatePropertyPage;
var
  iCheck : Integer;
  pLink : IPropertyLink;
begin
  pLink := IUnknown(OleObject) as IPropertyLink;
  if (pLink <> nil) then begin
     pLink.get_Enabled(iCheck);
     if (iCheck <> 0) then
        IsEnabled.Checked := True
     else
        IsEnabled.Checked := False;
  end;
end;

procedure TAddInPropPage.UpdateObject;
var  pLink : IPropertyLink;
begin
  pLink := IUnknown(OleObject) as IPropertyLink;
  if (pLink <> nil) then begin
     if (IsEnabled.Checked = True) then
        pLink.put_Enabled(1)
     else
        pLink.put_Enabled(0);
  end;
end;
```

Свойство может быть доступно (из программы 1С:Предприятие) только для чтения, только для записи или как для чтения, так и для записи. Реализуются эти возможности в контексте методов `IsPropReadable` (листинг 5.24) и `IsPropWritable` (листинг 5.25).

**Листинг 5.24. Установка флага доступности (из программы 1С:Предприятие) свойств для чтения**

```
function TAddInObject.IsPropReadable(lPropNum: Integer;
                          var pboolPropRead: Integer): HResult; stdcall;
begin
```

```
  case TProperties(lPropNum) of
    propIsEnabled: // Значение перечисления TProperties
     begin
       pboolPropRead := 1; // Свойство доступно для чтения
     end;
    else
      begin
        IsPropReadable := S_FALSE;
        Exit;
      end;
  end;
  IsPropReadable := S_OK;
end;
```

***Примечание***

В версии 1С:Предприятие 7.7 объявленное свойство всегда доступно хотя бы для чтения.

**Листинг 5.25. Установка флага доступности (из программы 1С:Предприятие) свойств для записи**

```
function TAddInObject.IsPropWritable(lPropNum: Integer;
                          var pboolPropWrite: Integer): HResult; stdcall;
begin
  case TProperties(lPropNum) of
    propIsEnabled: // Значение перечисления TProperties
      begin
        pboolPropWrite := 1; // Свойство доступно для записи
      end;
    else
      begin
        IsPropWritable := S_FALSE;
        Exit;
      end;
  end;
  IsPropWritable := S_OK;
end;
```

В данной версии примера оба метода внешней компоненты `Включить` и `Выключить` вызываются из программы 1С:Предприятие как процедуры. Реализовано это в методе внешней компоненты `CallAsProc` (листинг 5.26).

**Листинг 5.26. Реализация вызова методов `Включить` и `Выключить` как процедур**

```
function TAddInObject.CallAsProc(lMethodNum: Integer;
                           var paParams: PSafeArray): HResult; stdcall;
begin
  case TMethods(lMethodNum) of
    methEnable: // Значение перечисления TMethods
      begin
        boolIsEnabled := 1; // Включить компоненту
      end;
    methDisable: // Значение перечисления TMethods
      begin
        boolIsEnabled := 0; // Выключить компоненту
      end;
    else
      begin
        CallAsProc := S_FALSE;
        Exit;
      end;
  end;
  CallAsProc := S_OK;
end;
```

Сохраним наш Delphi-проект и скомпилируем его. Перед компиляцией проекта нужно проследить, чтобы в свойствах проекта (команда **Project > Options**) на вкладке **Linker** была включена опция **Generate C object files**. Результат компиляции (файл AddIn_Example01_01.DLL) нужно скопировать в любой из каталогов:

- каталог пользователя;
- каталог информационной базы;
- каталог с исполняемым файлом системы 1С:Предприятие.

В данном случае лучше всего скопировать файл AddIn_Example01_01.DLL в каталог "нашей" конфгурации (1C_example.DB).

Теперь посмотрим, каким образом можно использовать свойства и методы внешней компоненты `AddIn_Example01` в среде программы 1С:Предприятие.

***Примечание***

Сначала внешнюю компоненту нужно подключить к программе 1С:Предприятие. Для этого есть два штатных метода встроенного языка программирования системы 1С:Предприятие — `ЗагрузитьВнешнююКомпоненту` и `ПодключитьВнешнююКомпоненту`. Подробную информацию об этих методах можно найти в специальной литературе по программе 1С:Предприятие.

***Примечание***

Внешняя компонента может быть подключена как в контексте глобального модуля, так и в любой обработке (включая и внешнюю обработку). Мы рассмотрим вариант подключения внешней компоненты в контексте внешней обработки.

Запустим Конфигуратор системы 1С:Предприятие, откроем "нашу" конфигурацию и создадим внешнюю обработку (рис. 5.6).

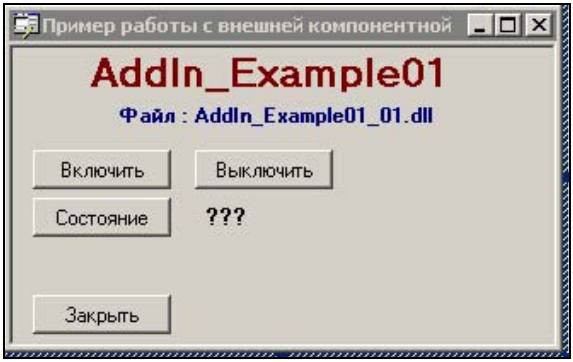



**Рис. 5.6.** Форма внешней обработки для тестирования внешней компоненты (файл AddIn_Example01_01.DLL)

Чтобы проверить функциональность внешней компоненты `AddIn_Example01`, реализованной в файле AddIn_Example01_01.DLL, достаточно трех кнопок (табл. 5.3).

***Таблица 5.3.*** *Назначение кнопок в форме внешней обработки*

| Кнопка | Назначение |
|---|---|
| **Включить** | Тестирование метода `Enable` (`Включить`) внешней компоненты |
| **Выключить** | Тестирование метода `Disable` (`Выключить`) внешней компоненты |
| **Состояние** | Тестирование свойства `IsEnabled` (`Включен`) внешней компоненты |

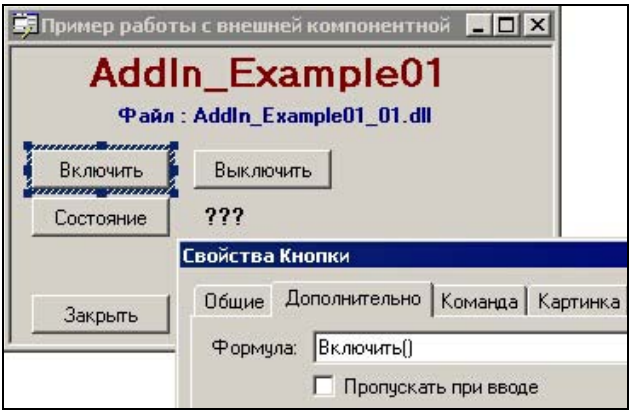



**Рис. 5.7.** Задание имени процедуры в качестве обработчика нажатия кнопки **Включить**

**Рис. 5.8.** Задание имени процедуры в качестве обработчика нажатия кнопки **Выключить**

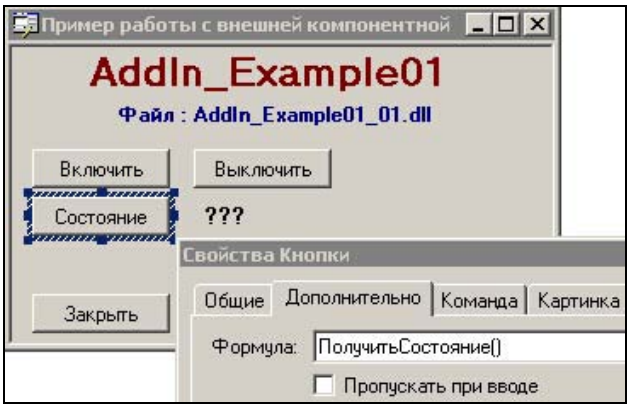



**Рис. 5.9.** Задание имени процедуры в качестве обработчика нажатия кнопки **Состояние**

"Привяжем" к каждой кнопке (табл. 5.6) процедуру-обработчик события "нажатие этой кнопки". Для этого нужно вызвать диалоговое окно настройки для каждой кнопки и на вкладке **Дополнительно** в поле **Формула** ввести имя соответствующей процедуры (рис. 5.7—5.9).

Каждая из указанных процедур должна присутствовать в тексте модуля обработки (листинг 5.27).

Далее, для настройки элемента управления `Текст` (надпись **???** на рис. 5.6) нужно вызвать соответствующее диалоговое окно и на вкладке **Общие** в поле **Идентификатор** ввести идентификатор этого элемента управления (рис. 5.10).

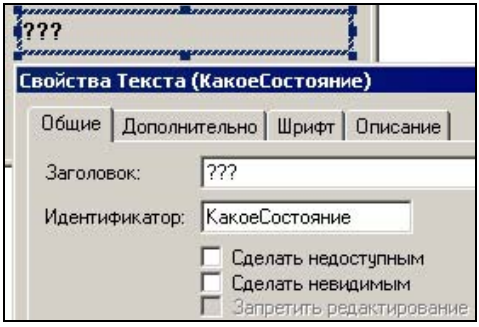



**Рис. 5.10.** Идентификатор элемента управления "Текст"

Для того чтобы можно было визуально отобразить состояние внешней компоненты (свойство `Включен`) в элементе управления `КакоеСостояние`, нужно на вкладке **Дополнительно** в поле **Формула** внести имя переменной (в данном случае `СостояниеОбъекта`, рис. 5.11), объявленной в тексте модуля обработки (листинг 5.27).

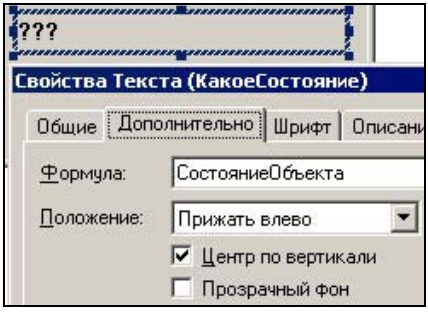



**Рис. 5.11.** "Привязка" элемента управления `КакоеСостояние` к переменной модуля `СостояниеОбъекта`

Теперь значение переменной `СостояниеОбъекта` будет отображаться в элементе управления `КакоеСостояние` автоматически.

Исходный текст модуля внешней обработки представлен в листинге 5.27.

**Листинг 5.27. Текст модуля внешней обработки ВнешняяФорма_для_AddIn_Example01_01.ERT**

```
//--------------------
// Переменная, к которой "привязан" элемент управления "КакоеСостояние"
Перем СостояниеОбъекта;
//--------------------
//--------------------
// Ссылка на внешнюю компоненту
Перем ВК Экспорт;
//--------------------
//--------------------
// Обработчик нажатия кнопки "Включить"
Процедура Включить()
  ВК.Включить();
КонецПроцедуры
//--------------------
//--------------------
//Обработчик нажатия кнопки "Выключить"
Процедура Выключить()
  ВК.Выключить();
КонецПроцедуры
//--------------------
//--------------------
//Обработчик нажатия кнопки "Состояние"
Процедура ПолучитьСостояние()
  // Читаем значение свойства внешней компоненты
  Если (ВК.Включен > 0) Тогда
    // Значение переменной "СостояниеОбъекта"
    // автоматически отображается в элементе
    // управления "КакоеСостояние"
    СостояниеОбъекта = "Состояние: Включено";
  Иначе
    // Значение переменной "СостояниеОбъекта"
    // автоматически отображается в элементе
```

```
    // управления "КакоеСостояние"
    СостояниеОбъекта = "Состояние: Выключено";
  КонецЕсли;
КонецПроцедуры
//-------------------
// Предопределенная процедура
Процедура ПриОткрытии()
  // Подключаем внешнюю компоненту
  ЗагрузитьВнешнююКомпоненту("AddIn_Example01_01.dll");
  // Получаем ссылку на OLE-объект внешней компоненты
  ВК = СоздатьОбъект("AddIn.AddIn_Example01");
  // Сразу же считываем состояние внешней компоненты
  ПолучитьСостояние();
КонецПроцедуры
```

Теперь сохраним обработку в файле с именем ВнешняяФорма_для_AddIn_Example01_01.ERT, запустим программу 1С:Предприятие и откроем сохраненную внешнюю обработку (рис. 5.12).

**Рис. 5.12.** Внешняя обработка для тестирования внешней компоненты `AddIn_Example01` (версия 1)

Чтобы проверить, как функционирует внешняя компонента, нужно поочередно нажимать кнопки **Включить** и **Выключить**, проверяя с помощью кнопки **Состояние**, изменилось ли состояние внешней компоненты.

Если есть необходимость отсылки в 1С:Предприятие сообщения о возникновении какого-либо внешнего события, то можно использовать для этого методы интерфейса `IAsyncEvent` (листинг 5.28).

**Листинг 5.28. Интерфейс для отсылки сообщения в 1С:Предприятие о каком-либо событии**

```
IAsyncEvent = interface(IUnknown)
  ['{AB634004-F13D-11D0-A459-004095E1DAEA}']
  procedure SetEventBufferDepth(lDepth: Integer); safecall;
  procedure GetEventBufferDepth(var plDepth: Integer); safecall;
  procedure ExternalEvent(const bstrSource, bstrMessage,
                          bstrData: WideString); safecall;
  procedure CleanBuffer; safecall;
end;
```

В табл. 5.4 описано назначение методов интерфейса `IAsyncEvent`.

***Таблица 5.4.*** *Назначение методов интерфейса* `IAsyncEvent`

| Метод | Назначение |
|---|---|
| `SetEventBufferDepth` | Устанавливает размер очереди сообщений о внешних событиях |
| `GetEventBufferDepth` | Возвращает размер очереди сообщений о внешних событиях |
| `ExternalEvent` | Помещает в очередь сообщение о внешнем событии |
| `CleanBuffer` | Очищает очередь сообщений |

Рассмотрим эту возможность на примере внешней компоненты `AddIn_Example01`. Для этого модифицируем метод `CallAsProc` следующим образом (листинг 5.29).

**Листинг 5.29. Модификация метода `CallAsProc`. Возбуждение события об изменении состояния**

```
function TAddInObject.CallAsProc(lMethodNum: Integer;
    var paParams: PSafeArray{(OleVariant)}): HResult; stdcall;
begin
  case TMethods(lMethodNum) of
    methEnable:
      begin
        boolIsEnabled := 1;
        if (pEvent <> nil)then begin
           pEvent.ExternalEvent('ВК01','Ответ','1');
```

```
        end;
      end;
    methDisable:
      begin
        boolIsEnabled := 0;
        if (pEvent <> nil)then begin
           pEvent.ExternalEvent('BK01','Ответ','0');
        end;
      end;
    else
      begin
        CallAsProc := S_FALSE;
        Exit;
      end;
  end;
  CallAsProc := S_OK;
end;
```

В данном случае в текст реализации метода `CallAsProc` добавлены строки, обеспечивающие отсылку в программу 1С:Предприятие сообщения об изменении состояния внешней компоненты (свойство `Включен`). Для этого вызывается метод `ExternalEvent` интерфейса `IAsyncEvent` (`pEvent`). Первый параметр, передаваемый методу `ExternalEvent`, это имя источника события (сообщения), второй параметр — имя сообщения, третий параметр — данные.

Сообщение помещается в очередь сообщений. В системе 1С:Предприятие есть специальная предопределенная процедура, которая автоматически (если она реализована в программном модуле) вызывается для обработки очереди сообщений.

Теперь нужно сохранить проект, скомпилировать его и результат компиляции (DLL-файл) скопировать в каталог 1C_example.DB.

***Примечание***

На сопроводительном компакт-диске книги исходные тексты этой версии примера находятся в папке \Part_1\Chapter_05\Topic_5_2\AddIn_Example01_02.PAS.

Теперь модифицируем внешнюю обработку (программы 1С:Предприятие) следующим образом. Добавим в форму еще два элемента управления типа "Текст" (рис. 5.13).

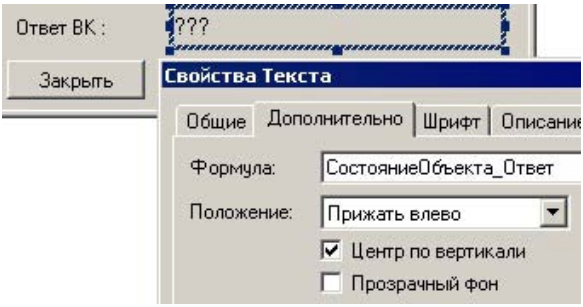



**Рис. 5.13.** Внешняя обработка для тестирования внешней компоненты `AddIn_Example01` (версия 2)

Здесь `СостояниеОбъекта_Ответ` — переменная, объявленная в тексте модуля обработки (листинг 5.30). Ее значение меняется в предопределенной процедуре `ОбработкаВнешнегоСобытия`.

**Листинг 5.30. Модуль внешней обработки ВнешняяФорма_для_AddIn_Example01_02.ERT**

```
Перем ВК Экспорт;
Перем СостояниеОбъекта;
Перем СостояниеОбъекта_Ответ;

Процедура Включить()
  ВК.Включить();
КонецПроцедуры

Процедура Выключить()
  ВК.Выключить();
КонецПроцедуры

Процедура ПолучитьСостояние()
  Если (ВК.Включен > 0) Тогда
    СостояниеОбъекта = "Состояние: Включено";
  Иначе
    СостояниеОбъекта = "Состояние: Выключено";
  КонецЕсли;
КонецПроцедуры

//-------------------------------
```

```
// Предопределенная обработка сообщений о внешних событиях
Процедура ОбработкаВнешнегоСобытия(Источник,Событие,Данные)
  Если (Источник = "ВК01") И (Событие = "Ответ") Тогда
    Если (Данные = "1") Тогда
      СостояниеОбъекта_Ответ =  "Состояние: Включено";
    Иначе
      СостояниеОбъекта_Ответ =  "Состояние: Выключено";
    КонецЕсли
  КонецЕсли;
КонецПроцедуры
//-------------------------------

Процедура ПриОткрытии()
  СостояниеОбъекта_Ответ = "???";
  ЗагрузитьВнешнююКомпоненту("AddIn_Example01_02.dll");
  ВК = СоздатьОбъект("AddIn.AddIn_Example01");
  ПолучитьСостояние();
КонецПроцедуры
```

Здесь видно, что сообщения, выбираемые из очереди сообщений (процедура `ОбработкаВнешнегоСобытия`), фильтруются по источнику и названию сообщения. В зависимости от значения параметра `Данные` переменная `СостояниеОбъекта_Ответ` принимает соответствующее значение.

Сохраним обработку под именем ВнешняяФорма_для_AddIn_Example01_02.ERT, запустим программу 1С:Предприятие и откроем сохраненную обработку (рис. 5.14).

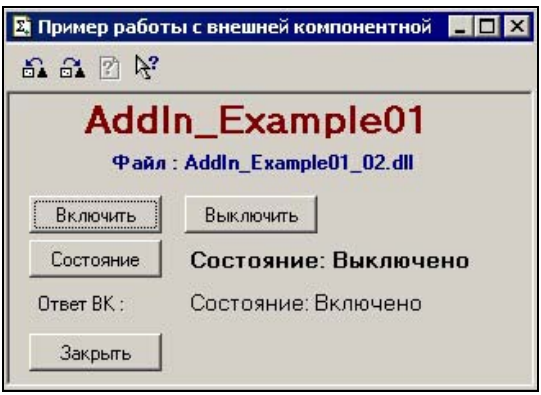



**Рис. 5.14.** Внешняя обработка для тестирования внешней компоненты `AddIn_Example01` (версия 2). Доработка формы

Теперь, нажимая на кнопки **Включить** и **Выключить**, можно убедиться, что механизм сообщений о внешних событиях функционирует.

Рассмотрим еще один пример, связанный с механизмом сообщений о внешних событиях. Еще раз модифицируем "нашу" внешнюю компоненту.

> ***Примечание***
>
> На сопроводительном компакт-диске книги исходные тексты этой версии примера находятся в папке \Part_1\Chapter_05\Topic_5_2\AddIn_Example01_03.PAS.

Добавим возможность генерировать сообщения для 1С:Предприятие, связанные с таймером. Для этого объявим еще одно свойство объекта `TAddInObject` — `TimerAnswer` типа `TTimer` и метод `OnMyTimerAnswer` типа `TNotifyEvent` (листинг 5.31).

**Листинг 5.31. Новые свойство и метод объекта `TAddInObject`**

```
TAddInObject = class ( ... )
   boolIsEnabled : Integer;
   TimerAnswer : TTimer;
   ...
   procedure OnMyTimerAnswer(Sender: TObject);
   ...
end;
```

Внесем соответствующие изменения в методы `Init` и `Done` (листинг 5.32).

**Листинг 5.32. Изменения в методах `Init` и `Done` объекта `TAddInObject`**

```
function TAddInObject.Init(pConnection: IDispatch): HResult; stdcall;
var
  ...
begin
  ...
  TimerAnswer := TTimer.Create(NIL);
  TimerAnswer.Interval:=55;
  TimerAnswer.Enabled := False;
  TimerAnswer.OnTimer := OnMyTimerAnswer;
  ...
end;
```

```
function TAddInObject.Done: HResult; stdcall;
begin
  ...
  TimerAnswer.Destroy();
  ...
end;
```

В листинге 5.33 представлен текст метода `OnMyTimerAnswer`.

**Листинг 5.33. Метод `OnMyTimerAnswer` объекта `TAddInObject`**

```
procedure TAddInObject.OnMyTimerAnswer(Sender: TObject);
begin
  TimerAnswer.Enabled:=FALSE;
  if (pEvent <> nil)then begin
     if boolIsEnabled=0 then begin
        pEvent.ExternalEvent('BK01','ОтветПоТаймеру','0');
     end
     else begin
        pEvent.ExternalEvent('BK01','ОтветПоТаймеру','1');
     end;
  end;
end;
```

Как видите, здесь проверяется свойство `boolIsEnabled` и, в зависимости от значения свойства, генерируется соответствующее сообщение.

Внесем также измнения в текст метода `CallAsProc` (листинг 5.34).

**Листинг 5.34. Изменения в тексте метода `CallAsProc` объекта `TAddInObject`**

```
function TAddInObject.CallAsProc(lMethodNum: Integer;
     var paParams: PSafeArray): HResult; stdcall;
begin
  case TMethods(lMethodNum) of
    //----------------------------------------
    methEnable:
      begin
        boolIsEnabled := 1;
        if (pEvent <> nil)then begin
```

```
            pEvent.ExternalEvent('ВК01','Ответ','1');
          end;
          TimerAnswer.Enabled:=TRUE;
        end;
      methDisable:
        begin
          boolIsEnabled := 0;
          if (pEvent <> nil)then begin
            pEvent.ExternalEvent('ВК01','Ответ','0');
          end;
          TimerAnswer.Enabled:=TRUE;
        end;
      //----------------------------------------
      else
        begin
          CallAsProc := S_FALSE;
          Exit;
        end;
    end;
    CallAsProc := S_OK;
end;
```

Идея заключается в том, что когда из программы 1С:Предприятие вызывается какой-либо из методов внешней компоненты `Включить` или `Включить`, то наряду с рассмотренным ранее вариантом генерации соответствующего сообщения запускается и таймер. Через 55 мс срабатывает таймер и, в зависимости от значения `boolIsEnabled`, генерируется еще одно сообщение (листинг 5.33).

Сохраним наш Delphi-проект под именем AddIn_Example01_03.DPR, скомпилируем его и скопируем файл AddIn_Example01_03.DLL в папку 1C_example.DB.

Теперь вновь запустим Конфигуратор 1С:Предприятие, откроем внешнюю обработку предыдущего примера и сделаем некоторые изменения.

Добавим в форму внешней обработки элемент управления "список" (рис. 5.15).

Изменим имя DLL-файла внешней компоненты, который нужно загрузить (листинг 5.35).

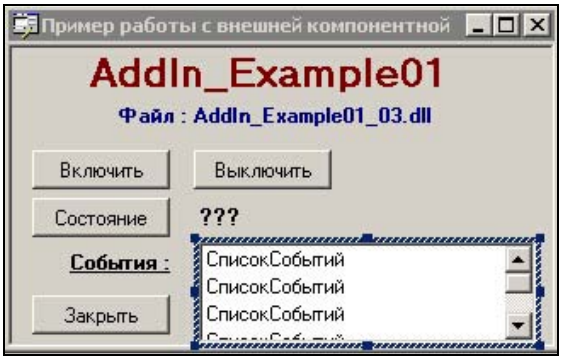



**Рис. 5.15.** Внешняя обработка для тестирования внешней компоненты `AddIn_Example01` (версия 3)

**Листинг 5.35. Процедура `ПриОткрытии` внешней обработки 1С:Предприятие**

```
Процедура ПриОткрытии()
  СостояниеОбъекта_Ответ = "???";
  ЗагрузитьВнешнююКомпоненту("AddIn_Example01_03.dll");
  ВК = СоздатьОбъект("AddIn.AddIn_Example01");
  ПолучитьСостояние();
КонецПроцедуры
```

Изменим текст процедуры `ОбработкаВнешнегоСобытия` следующим образом (листинг 5.36).

**Листинг 5.36. Измененный текст процедуры `ОбработкаВнешнегоСобытия`**

```
Процедура ОбработкаВнешнегоСобытия(Источник,Событие,Данные)

 СписокСобытий.ДобавитьЗначение(СокрЛП(Источник)
               +"."+СокрЛП(Событие)
               +":"+СокрЛП(Данные));

Если (Источник = "ВК01") И (Событие = "Ответ") Тогда
  Если (Данные = "1") Тогда
   СостояниеОбъекта_Ответ = "Состояние: Включено";
  Иначе
```

```
    СостояниеОбъекта_Ответ = "Состояние: Выключено";
  КонецЕсли
 КонецЕсли;
КонецПроцедуры
```

То есть теперь все сообщения о внешних событиях будут отражены в элементе управления `СписокСобытий`.

Сохраним обработку под именем ВнешняяФорма_для_AddIn_Example01_03.ERT, запустим программу 1С:Предприятие и откроем сохраненную обработку.

На рис. 5.16 представлен результат поочередного нажатия кнопок **Включить** и **Выключить**.

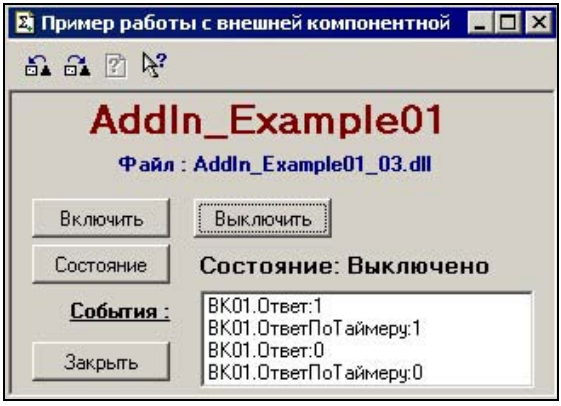



**Рис. 5.16.** Внешняя обработка для тестирования внешней компоненты `AddIn_Example01` (версия 3). Доработка формы

Теперь расширим функционал внешней компоненты. За основу возьмем версию 2 "нашей" внешней компоненты.

> ***Примечание***
>
> На сопроводительном компакт-диске книги исходные тексты новой (четвертой) версии примера находятся в папке \Part_1\Chapter_05\Topic_5_2\AddIn_Example01_04.PAS.

Добавим несколько новых процедур и функций для 1С:Предприятие (табл. 5.5 и листинг 5.37).

***Таблица 5.5.*** *Новые методы внешней компоненты*

| Метод | Назначение |
|---|---|
| `ВывестиВСтрокуСтатуса` (или `OutputToStatusLine`) | Выводит сообщение в строку состояния (статуса) программы 1С:Предприятие. Аналог штатной процедуры `Состояние()`. Синтаксис: `ВывестиВСтрокуСтатуса(`*`Строка, Задержка`*`);` *`Задержка`* — время, на которое выведенное в строку состояния сообщение "заморозится" (в секундах) |
| `Сообщить` (или `Message`) | Выводит строку в окно сообщений программы 1С:Предприятие. Упрощенный аналог штатной процедуры `Сообщить()`. Синтаксис: `Сообщить(`*`Строка`*`);` |
| `ОчиститьОкноСообщений` (или `ClearMessageWindow`) | Очищает окно сообщений программы 1С:Предприятие. Аналог штатной процедуры `ОчиститьОкноСообщений()`. Синтаксис: `ОчиститьОкноСообщений ();` |
| `ПолучитьСписокФайлов` (или `GetListFiles`) | Получить список файлов по маске. В качестве параметра принимает `СписокЗначений`, где размещает список найденных файлов. Синтаксис: `ПолучитьСписокФайлов (`*`СписЗнач, МаскаФайлов`*`);` Параметры: *`СписЗнач`* — `СписокЗначений`, должен быть создан в программном модуле 1С:Предприятие; *`МаскаФайлов`* — может включать путь к файлам, например, `C:\MyDir\*.txt`. Возвращаемое значение — количество найденных файлов |
| `СоздатьСписокФайлов` (или `CreateListFiles`) | Создает список файлов по маске: создает `СписокЗначений`, размещает в нем список найденных файлов и возвращает его в вызвавший программный модуль 1С:Предприятие. Синтаксис: `СоздатьСписокФайлов (`*`МаскаФайлов`*`);` *`МаскаФайлов`* может включать путь к файлам, например, `C:\MyDir\*.txt` |

**Листинг 5.37. Имена для новых процедур и функций (для доступа в программе 1С:Предприятие)**

```
resourcestring
  strIsEnabled = 'IsEnabled,Включен';
  strEnable = 'Enable,Включить';
```

```
  strDisable = 'Disable,Выключить';
  strOutputToStatusLine = 'OutputToStatusLine,ВывестиВСтрокуСтатуса';
  strMessage = 'Message,Сообщить';
  strClearMessageWindow = 'ClearMessageWindow,ОчиститьОкноСообщений';
  strGetListFiles  = 'CreateListFiles,СоздатьСписокФайлов';
  strGetListFiles2 = 'GetListFiles,ПолучитьСписокФайлов';
```

Добавим новые значения перечисления `TMethods` (для реализации указанной выше функциональности).

**Листинг 5.38. Новые значения перечисления `TMethods`**

```
type
 TMethods = ( methEnable,
              methDisable,
              methOutputToStatusLine, // ВывестиВСтрокуСтатуса
              methMessage,            // Сообщить
              methClearMessageWindow, // ОчиститьОкноСообщений
              methGetListFiles,       // СоздатьСписокФайлов
              methGetListFiles2,      // ПолучитьСписокФайлов
              LastMethod );
```

***Примечание***

Обратите внимание: значение `LastMethod` всегда должно оставаться последним!

Добавим строку `pConn : IDispatch;` в объявление объекта `TAddInObject` (листинг 5.39).

**Листинг 5.39. Новое свойство объекта `TAddInObject`**

```
TAddInObject = class ( ... )
   boolIsEnabled : Integer;
   pConn : IDispatch;
   ...
end;
```

В метод `Init` объекта `TAddInObject` добавим строку, где свойству `pConn` будет присвоен указатель на интерфейс `IDispatch` (листинг 5.40).

**Листинг 5.40. Дополнение в тексте метода `Init` объекта `TAddInObject`**

```
function TAddInObject.Init(pConnection: IDispatch): HResult; stdcall;
var
  iRes : Integer;
begin
  // Сохраним указатель на интерфейс IDispatch
  pConn := pConnection;
   ...
end;
```

Дополним текст метода `FindMethod` объекта `TAddInObject` следующим образом (листинг 5.41).

**Листинг 5.41. Дополненный текст метода `FindMethod` объекта `TAddInObject`**

```
function TAddInObject.FindMethod(const bstrMethodName: WideString;
     var plMethodNum: Integer): HResult; stdcall;
begin
  plMethodNum := -1;
  //----------------------------------------
  if (bstrMethodName = TermString(strEnable,0)) then plMethodNum := 0;
  if (bstrMethodName = TermString(strEnable,1)) then plMethodNum := 0;
  if (bstrMethodName = TermString(strDisable,0)) then plMethodNum := 1;
  if (bstrMethodName = TermString(strDisable,1)) then plMethodNum := 1;
  // ВывестиВСтрокуСтатуса
  if (bstrMethodName = TermString(strOutputToStatusLine,0))
    then plMethodNum := 2;
  if (bstrMethodName = TermString(strOutputToStatusLine,1))
    then plMethodNum := 2;
  // Сообщить
  if (bstrMethodName = TermString(strMessage,0)) then plMethodNum := 3;
  if (bstrMethodName = TermString(strMessage,1)) then plMethodNum := 3;
  // ОчиститьОкноСообщений
  if (bstrMethodName = TermString(strClearMessageWindow,0))
    then plMethodNum := 4;
  if (bstrMethodName = TermString(strClearMessageWindow,1))
    then plMethodNum := 4;
```

```
  // СоздатьСписокФайлов
  if (bstrMethodName = TermString(strGetListFiles,0))
    then plMethodNum := 5;
  if (bstrMethodName = TermString(strGetListFiles,1))
    then plMethodNum := 5;
  // ПолучитьСписокФайлов
  if (bstrMethodName = TermString(strGetListFiles2,0))
    then plMethodNum := 6;
  if (bstrMethodName = TermString(strGetListFiles2,1))
    then plMethodNum := 6;
  //----------------------------------------
  if (plMethodNum = -1) then begin
      FindMethod := S_FALSE;
      Exit;
  end;
  FindMethod := S_OK;
end;
```

Дополним текст метода `GetMethodName` объекта `TAddInObject` следующим образом (листинг 5.42).

**Листинг 5.42. Дополненный текст метода `GetMethodName` объекта `TAddInObject`**

```
function TAddInObject.GetMethodName(lMethodNum, lMethodAlias: Integer;
    var pbstrMethodName: WideString): HResult; stdcall;
begin
  pbstrMethodName := '';
  case TMethods(lMethodNum) of
    methEnable:
      begin
        pbstrMethodName := TermString(strEnable,lMethodAlias);
      end;
    methDisable:
      begin
        pbstrMethodName := TermString(strDisable,lMethodAlias);
      end;
    methOutputToStatusLine: // ВывестиВСтрокуСтатуса
      begin
```

```
        pbstrMethodName :=
                TermString(strOutputToStatusLine,lMethodAlias);
      end;
    methMessage: // Сообщить
      begin
        pbstrMethodName := TermString(strMessage,lMethodAlias);
      end;
    methClearMessageWindow: // ОчиститьОкноСообщений
      begin
        pbstrMethodName :=
                TermString(strClearMessageWindow,lMethodAlias);
      end;
    methGetListFiles:  // СоздатьСписокФайлов
      begin
        pbstrMethodName := TermString(strGetListFiles,lMethodAlias);
      end;
    methGetListFiles2:  // ПолучитьСписокФайлов
      begin
        pbstrMethodName := TermString(strGetListFiles2,lMethodAlias);
      end;
    else
      begin
        GetMethodName := S_FALSE;
        Exit;
    end;
  end;
  GetMethodName := S_OK;
end;
```

До этого момента ничего нового (относительно предыдущих версий примера) не было. Но теперь мы создаем процедуры и функции, которые имеют входные параметры. Это должно быть отражено в методе `GetNParams` объекта `TAddInObject` соответствующим образом (листинг 5.43).

**Листинг 5.43. Метод `GetNParams` объекта `TAddInObject`**

```
function TAddInObject.GetNParams(lMethodNum: Integer;
                          var plParams: Integer): HResult; stdcall;
begin
```

```
  plParams := 0;
  case TMethods(lMethodNum) of
    methEnable:
      begin
        plParams := 0;
      end;
    methDisable:
      begin
        plParams := 0;
      end;
    methOutputToStatusLine: // ВывестиВСтрокуСтатуса(Строка,Задержка)
      begin
        plParams := 2;
      end;
    methMessage: // Сообщить(Строка)
      begin
        plParams := 1;
      end;
    methClearMessageWindow: // ОчиститьОкноСообщений()
      begin
        plParams := 0;
      end;
    methGetListFiles: // СоздатьСписокФайлов(МаскаФайлов)
      begin
        plParams := 1;
      end;
    methGetListFiles2: //ПолучитьСписокФайлов(СписокЗначений,МаскаФайлов)
      begin
        plParams := 2;
      end;
    else
      begin
        GetNParams := S_FALSE;
        Exit;
    end;
  end;
  GetNParams := S_OK;
end;
```

В данном примере методы `СоздатьСписокФайлов` и `ПолучитьСписокФайлов` реализованы как функции (при их вызове из программных модулей 1С:Предприятие). Это должно быть отражено в методе `HasRetVal` объекта `TAddInObject` следующим образом (листинг 5.44).

**Листинг 5.44. Метод `HasRetVal` объекта `TAddInObject`**

```
function TAddInObject.HasRetVal(lMethodNum: Integer;
                          var pboolRetValue: Integer): HResult; stdcall;
begin
  pboolRetValue := 0;
  case TMethods(lMethodNum) of
    methGetListFiles:  // СоздатьСписокФайлов(МаскаФайлов)
      begin
        pboolRetValue := -1;
      end;
    methGetListFiles2:  // ПолучитьСписокФайлов
      begin
        pboolRetValue := -1;
      end;
    else
      begin
        HasRetVal := S_FALSE;
        Exit;
      end;
  end;
  HasRetVal := S_OK;
end;
```

***Примечание***

В OLE Automation логическое значение `TRUE` соответствует целочисленному значению –1, а значение `FALSE` — целочисленному значению 0.

Теперь, собственно, нужно написать код реализации функционала внешней компоненты. При этом процедуры должны быть реализованы в методе `CallAsProc`, а функции — в методе `CallAsFunc` объекта `TAddInObject` (листинги 5.45 и 5.47).

**Листинг 5.45. Метод `CallAsProc` объекта `TAddInObject`**

```
function TAddInObject.CallAsProc(lMethodNum: Integer;
                          var paParams: PSafeArray): HResult; stdcall;
Var
 List  : TStrings;
 sMask : String;
 V0    : Variant;
 V     : Variant;
 i,Res : Integer;
begin
  case TMethods(lMethodNum) of
    ...
    methOutputToStatusLine: // ВывестиВСтрокуСтатуса(Строка,Задержка)
      begin
        pStatusLine.SetStatusLine(GetNParam(paParams,0));
        if GetNParam(paParams,1)*1>0 then begin
           Sleep(GetNParam(paParams,1)*1000);
        end;
      end;
    methMessage: // Сообщить(Строка)
      begin
        V := pConn;
        // Получаем указатель на интерфейс
        // IDispatch программы 1С:Предприятие
        V := V.AppDispatch;
        // Увеличиваем счетчик ссылок на объект интерфейса
        IDispatch(V)._AddRef;
        // Вызываем штатную процедуру Сообщить()
        // с помощью метода ExecuteBatch
        V.ExecuteBatch('Сообщить("'+TrimRight(GetNParam(paParams,0))
                                  +'");');
      end;
    methClearMessageWindow: // ОчиститьОкноСообщений()
      begin
        V := pConn;
        // Получаем указатель на интерфейс
        // IDispatch программы 1С:Предприятие
        V := V.AppDispatch;
        // Увеличиваем счетчик ссылок на объект интерфейса
```

```
        IDispatch(V)._AddRef;
        // Вызываем штатную процедуру 'ОчиститьОкноСообщений()
        // с помощью метода ExecuteBatch
        V.ExecuteBatch('ОчиститьОкноСообщений();');
      end;
    else
      begin
        CallAsProc := S_FALSE;
        Exit;
      end;
  end;
  CallAsProc := S_OK;
end;
```

Здесь требуются некоторые пояснения.

При реализации процедуры `ВывестиВСтрокуСтатуса` был использован метод `SetStatusLine` интерфейса `IStatusLine` (`pStatusLine`), листинг 5.46.

**Листинг 5.46. Объявление интерфейса `IStatusLine` (модуль AddInLib.PAS)**

```
IStatusLine = interface(IUnknown)
  ['{AB634005-F13D-11D0-A459-004095E1DAEA}']
  procedure SetStatusLine(const bstrSource: WideString); safecall;
  procedure ResetStatusLine; safecall;
end;
```

Для реализации процедур `Сообщить` и `ОчиститьОкноСообщений` используется метод `ExecuteBatch` программы 1С:Предприятие (который был подробно рассмотрен в *разд. 4.6*), с помощью которого вызываются соответствующие штатные процедуры программы 1С:Предприятие (более подробно об этих функциях можно прочитать в специальной литературе по этой системе).

Теперь нужно написать код, реализующий функции `СоздатьСписокФайлов` и `ПолучитьСписокФайлов` (листинг 5.47).

**Листинг 5.47. Метод `CallAsFunc` объекта `TAddInObject`**

```
function TAddInObject.CallAsFunc(lMethodNum: Integer;
                         var pvarRetValue: OleVariant;
                         var paParams: PSafeArray): HResult; stdcall;
```

```
Var
 List : TStrings;
 sMask: string;
 V0   : OleVariant;
 V    : Variant;
 i    : Integer;
begin
  case TMethods(lMethodNum) of
    //----------------------------------------
    methGetListFiles: // СоздатьСписокФайлов
      begin
        List := TStringList.Create;
        TRY
          V:=pConn;
          // Получаем указатель на интерфейс
          // IDispatch программы 1С:Предприятие
          V := V.AppDispatch;
          // Увеличиваем счетчик ссылок на объект интерфейса
          IDispatch(V)._AddRef;
          // Возвращаемое функцией значение
          // должно иметь агрегатный тип: "СписокЗначений".
          // Поэтому создаем "СписокЗначений"
          // с помощью метода EvalExpr.
          pvarRetValue:=V.EvalExpr('СоздатьОбъект("СписокЗначений")');
          // Получаем маску файлов из первого (и единственного)
          // переданного параметра
          sMask:=Trim(GetNParam(paParams,0));
          // Если маска не задана, выбираем все файлы
          if length(sMask)<=0 then sMask:='*.*';
          // Вызываем нашу функцию, которая отбирает
          // файлы по маске и возвращает их в List:TStrings
          sp_GetListFiles_Mask(sMask,List);
          if List.Count>0 then begin
             // Если список файлов не пуст,
             // переписываем имена файлов построчно
             // в созданный нами же "СписокЗначений"
             for i:=0 to (List.Count-1) do
              begin
               // С помощью метода "ДобавитьЗначение"
```

```
            // объекта "СписокЗначений"
            pvarRetValue.ДобавитьЗначение(List[i],List[i]);
          end;
        end;
      FINALLY
        // Очищаем "захваченные" нами ресурсы памяти
        if List<>NIL then begin
          List.Clear;
          List.Free;
          List:=NIL;
        end;
      END;
    end;
  methGetListFiles2: // ПолучитьСписокФайлов
    begin
      List := TStringList.Create;
      TRY
        // Возвращаемое функцией значение
        // (количество файлов)
        pvarRetValue:=0;
        // Получаем значение первого переданного параметра.
        // это должен быть объект типа: СписокЗначений
        V0:=GetNParam(paParams,0);
        // Проверяем, относится ли это значение к агрегатному типу
        if VarType(V0) = varDispatch then begin
          // Для простоты текста: не уточняем конкретный
          // тип объекта и предполагаем,
          // что это обязательно "СписокЗначений".
          // Очищаем список значений
          V0.УдалитьВсе;
          // Получаем маску файлов из второго
          // переданного параметра
          sMask:=Trim(GetNParam(paParams,1));
          // Если маска не задана, выбираем все файлы
          if length(sMask)<=0 then sMask:='*.*';
          // Вызываем нашу функцию, которая отбирает
          // файлы по маске и возвращает их в List:TStrings
          sp_GetListFiles_Mask(sMask,List);
          if List.Count>0 then begin
```

```
                // Если список файлов не пуст,
                // переписываем имена файлов построчно
                // в "СписокЗначений"
                for i:=0 to (List.Count-1) do
                 begin
                  // С помощью метода "ДобавитьЗначение"
                  // объекта "СписокЗначений"
                  V0.ДобавитьЗначение(List[i],List[i]);
                  // Счетчик файлов (возвращаемое значение)
                  pvarRetValue:=pvarRetValue+1;
                end;
             end;
             // Возвращаем "СписокЗначений" обратно
             // в программный модуль 1С:Предприятие
             PutNParam(paParams,0,V0);
          end;
        FINALLY
          // Очищаем "захваченные" нами ресурсы памяти
          if List<>NIL then begin
             List.Clear;
             List.Free;
             List:=NIL;
          end;
        END;
      end;
    //----------------------------------------
    else
      begin
        CallAsFunc := S_FALSE;
        Exit;
      end;
  end;
  CallAsFunc := S_OK;
end;
```

Листинг 5.47 содержит подробный комментарий. Текст функции `sp_GetListFiles_Mask` здесь не приводится из-за ее тривиальности — достаточно отметить, что там используются "стандартные" функции `SysUtils.FindFirst()`, `SysUtils.FindNext()`, `SysUtils.FindClose()`.

Сохраним наш Delphi-проект AddIn_Example01_04.DPR, скомпилируем его и скопируем файл AddIn_Example01_04.DLL в папку 1C_example.DB.

Теперь запустим Конфигуратор 1С:Предприятие и создадим новую внешнюю обработку ВнешняяФорма_для_AddIn_Example01_04.ERT (рис. 5.17).

Для начала протестируем созданные процедуры: `ВывестиВСтрокуСтатуса()`, `Сообщить()`, `ОчиститьОкноСообщений()`.

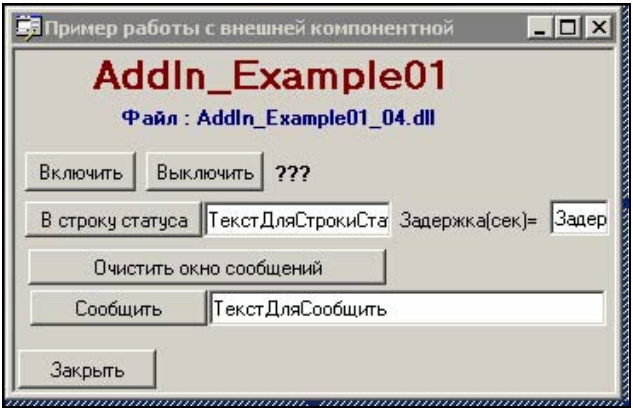



**Рис. 5.17.** Внешняя обработка для тестирования внешней компоненты `AddIn_Example01` (версия 4)

***Примечание***

Реализация функциональности кнопок **Включить** и **Выключить** подробно рассмотрена в предыдущих примерах.

Реакция на событие "нажатие кнопки" для кнопок **В строку статуса**, **Очистить окно сообщений** и **Сообщить** организована с помощью процедур `AddIn_ВывВСтрСтат()`, `AddIn_ОчиститьПанельСообщений()`, `AddIn_Сообщить()`.

В листинге 5.48 представлен текст модуля обработки ВнешняяФорма_для_AddIn_Example01_04.ERT.

**Листинг 5.48. Модуль внешней обработки ВнешняяФорма_для_AddIn_Example01_04.ERT**

```
Перем ВК Экспорт;
Перем СостояниеОбъекта;

Процедура Включить()
```

```
  ВК.Включить();
КонецПроцедуры

Процедура Выключить()
  ВК.Выключить();
КонецПроцедуры

Процедура AddIn_Сообщить()
  ВК.Сообщить(ТекстДляСообщить);
КонецПроцедуры

Процедура AddIn_ОчиститьПанельСообщений()
  ВК.ОчиститьОкноСообщений();
КонецПроцедуры

Процедура AddIn_ВывВСтрСтат()
  ВК.ВывестиВСтрокуСтатуса(ТекстДляСтрокиСтатуса,Задержка_сек);
КонецПроцедуры

Процедура ОбработкаВнешнегоСобытия(Источник,Событие,Данные)
  Если (Источник = "ВК01") И (Событие = "Включен") Тогда
    Если (Данные = "1") Тогда
      СостояниеОбъекта =  "Состояние: Включено";
    Иначе
      СостояниеОбъекта =  "Состояние: Выключено";
    КонецЕсли
  КонецЕсли;
КонецПроцедуры

Процедура ПриОткрытии()
  СостояниеОбъекта = "???";
  ТекстДляСтрокиСтатуса = "Привет, как дела ?";
  Задержка_сек = 7;
  ТекстДляСообщить = "Привет, как дела-2   ???";
  ЗагрузитьВнешнююКомпоненту("AddIn_Example01_04.dll");
  ВК = СоздатьОбъект("AddIn.AddIn_Example01");
КонецПроцедуры
```

На рис. 5.18 представлен результат тестирования процедур внешней компоненты.

Для того чтобы протестировать функции `СоздатьСписокФайлов` и `ПолучитьСписокФайлов`, внесем изменения во внешнюю обработку (рис. 5.19).

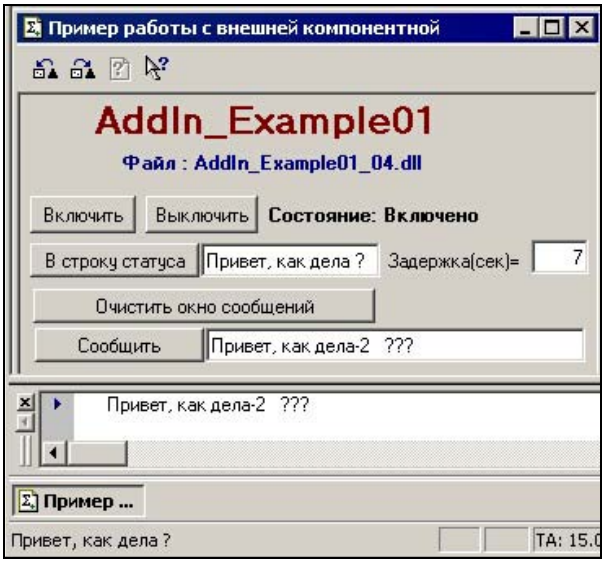



**Рис. 5.18.** Результат тестирования процедур внешней компоненты `AddIn_Example01` (версия 4)

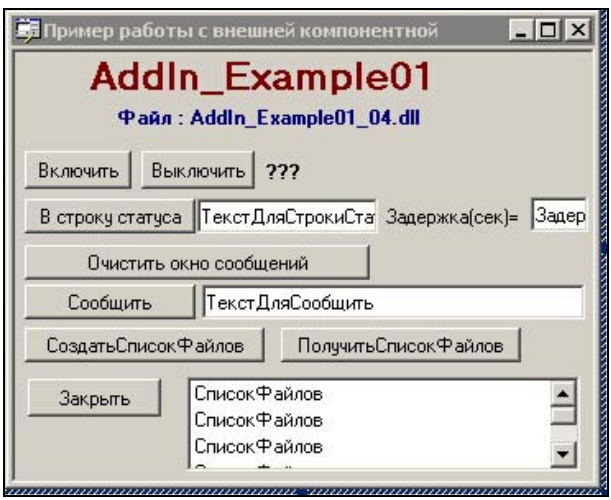



**Рис. 5.19.** Внешняя обработка `AddIn_Example01` (версия 4) для тестирования функций `СоздатьСписокФайлов` и `ПолучитьСписокФайлов`

В листинге 5.49 представлен текст процедур-обработчиков событий нажатия кнопок **СоздатьСписокФайлов** и **ПолучитьСписокФайлов** (рис. 5.19).

**Листинг 5.49. Обработчики событий нажатия кнопок (модуль внешней обработки ВнешняяФорма_для_AddIn_Example01_04.ERT)**

```
Процедура AddIn_СоздатьСписокФайлов()
  СписокФайлов.УдалитьВсе();
  Список = ВК.СоздатьСписокФайлов(КаталогИБ()+"*.*");
  X="";
  Список.ВыбратьЗначение(X);
КонецПроцедуры

Процедура AddIn_ПолучитьСписокФайлов()
  ВК.ПолучитьСписокФайлов(СписокФайлов,КаталогИБ()+"*.*");
КонецПроцедуры
```

На рис. 5.20 и 5.21 представлены результаты тестирования функций внешней компоненты `СоздатьСписокФайлов` и `ПолучитьСписокФайлов` соответственно.

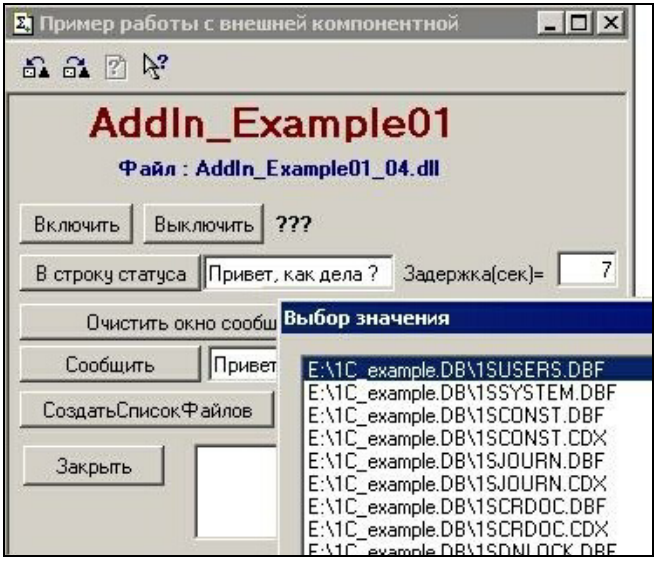



**Рис. 5.20.** Результат тестирования функции `СоздатьСписокФайлов`

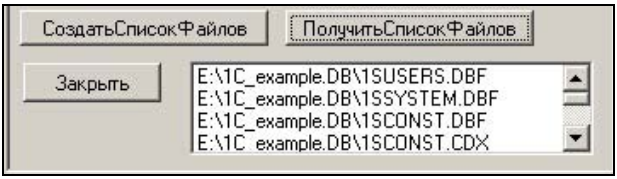



**Рис. 5.21.** Результат тестирования функции `ПолучитьСписокФайлов`

### *Примечание*

Рассмотренные примеры внешней компоненты `AddIn_Example01` носили иллюстративный характер и вряд ли могут быть полезны для практических целей.

Чтобы как-то компенсировать этот момент, на сопроводительном компакт-диске книги в папке \Part_1\Chapter_05\Topic_5_2\AddIn_Files.PAS приведены исходные тексты внешней компоненты `AddIn_Files`, функциональность которой ориентирована на работу с файловой системой и включает процедуры и функции (табл. 5.6), из которых не все есть среди методов объекта `ФС` системы 1С:Предприятие.

***Таблица 5.6.*** *Процедуры и функции внешней компоненты* `AddIn_Files`

| **Метод** | **Назначение** |
|---|---|
| Функция `ПолучитьНовоеУникальноеИмяФайла` (или `GetNewFileName`) | Получает на входе имя файла (включая полный путь) и проверяет, есть ли такой файл на диске. Если есть, то вычисляется новое имя файла (к нему добавляется единица; расширение остается неизменным). И так до тех пор, пока не будет вычислено уникальное имя файла |
| Функция `МожноСохранятьНаЭтотДиск` (или `ItsNotReadOnlyDisk`) | Получает имя диска (полное имя файла или папки) и проверяет, можно ли сохранять файлы на указанном диске |
| Функция `ПолучитьСписокФайлов` (или `GetListFiles`) | Получает на входе маску файлов, атрибуты файлов и `СписокЗначений`; возвращает в `СписокЗначений` список найденных файлов |
| Функция `ПолучитьРазмерФайла` (или `GetFileSize`) | Получает на входе имя файла и определяет размер этого файла в байтах |

***Таблица 5.6*** *(продолжение)*

| Метод | Назначение |
|---|---|
| Функция `РазрезатьФайл` (или `CutFile`) | Разделяет файл на части. Получает на входе имя файла, имя папки назначения, максимальный размер частей файла и префикс расширения частей файла. Заданный файл "нарезается" на части заданного размера, которые размещаются в заданной папке. Если задан последний входной параметр функции (`СписокЗначений`), то список полученных частей будет размещен в элементе `СпискоЗначений` |
| Функция `СклеитьФайл` (или `StickFile`) | Позволяет "склеить" файл из отдельных частей. Получает на входе маску частей файла и полное имя результирующего (склеенного) файла. Из найденных по маске частей "собирается" файл с заданным именем. Если задан последний входной параметр функции (`СписокЗначений`), то список имен найденных файлов-частей будет размещен в элементе `СпискеЗначений` |
| Функция `ПолучитьЛогическоеЗначениеИзINIфайла` (или `GetBoolenFromINI`) | Считывает (и возвращает) из заданного INI-файла значение типа `Boolean` и конвертирует его в целое число (`TRUE` — в 1, `FALSE` — в 0) |
| Процедура `ЗаписатьЛогическоеЗначениеВINIфайл` (или `PutBoolenToINI`) | Записывает в заданный INI-файл целочисленное значение как значение типа `Boolean`. Значения больше 0 трактуются как `TRUE`, остальные — как `FALSE`. Проверяется возможность записи на диск |
| Функция `ПолучитьЦелоеЧислоИзINIфайла` (или `GetIntegerFromINI`) | Считывает (и возвращает) из заданного INI-файла значение типа `Integer` |

***Таблица 5.6*** *(окончание)*

| Метод | Назначение |
|---|---|
| Процедура `ЗаписатьЦелоеЧислоВINIфайл` (или `PutIntegerToINI`) | Записывает в заданный INI-файл целочисленное значение. Проверяется возможность записи на диск |
| Функция `ПолучитьСтрокуИзINIфайла` (или `GetStringFromINI`) | Считывает (и возвращает) из заданного INI-файла значение типа `String` |
| Процедура `ЗаписатьСтрокуВINIфайл` (или `PutStringToINI`) | Записывает в заданный INI-файл строку. Проверяется возможность записи на диск |
| Функция `ПолучитьВещественноеЧислоИзINIфайла` (или `GetRealFromINI`) | Считывает (и возвращает) из заданного INI-файла значение типа `Real` |
| Процедура `ЗаписатьВещественноеЧислоВINIфайл` (или `PutRealToINI`) | Записывает в заданный INI-файл вещественное число. Проверяется возможность записи на диск |

***Примечание***

Более подробное описание этой внешней компоненты содержится на сопроводительном компакт-диске книги в файле ReadMe_AddIn_Files.DOC из папки \Part_1\Chapter_05\Topic_5_2\AddIn_Files.PAS.

Здесь нужно отметить следующее. Процедуры и функции программного модуля встроенного языка системы 1С:Предприятие могут иметь параметры, значения которых устанавливаются по умолчанию. Поддержка этой возможности во внешней компоненте реализована в методе `GetParamDefValue` следующим образом (листинг 5.50).

**Листинг 5.50. Метод `GetParamDefValue` объекта `TAddInObject`**

```
function TAddInObject.GetParamDefValue(lMethodNum, lParamNum: Integer;
         var pvarParamDefValue: OleVariant): HResult; stdcall;
begin
  VarClear(pvarParamDefValue);
  case TMethods(lMethodNum) of
    //------------------------------
    // РазрезатьФайл(ИмяФайла,ИмяПапкиКуда,РазмерЧастей,
    //                ПрефиксРасширения="aif",СписокЧастейФайла="")
    methCutFile:
```

```
      begin
        // ПрефиксРасширения
        if lParamNum=3 then pvarParamDefValue := 'aif';
        // СписокЧастейФайла — может отсутствовать
        if lParamNum=4 then pvarParamDefValue := '';
      end;
    //-----------------------------
    //-----------------------------
    // СклеитьФайл(МаскаФайлов,ИмяСклеенногоФайла,
    //             СписокФайловИзКоторыхСобрали="")
    methStickFile:
      begin
        // СписокФайловИзКоторыхСобрали — может отсутствовать
        if lParamNum=2 then pvarParamDefValue := '';
      end;
    //-----------------------------
    //-----------------------------
    // ПолучитьСписокФайлов(СписокЗначений,Маска,
    //                      Скрытые=0,Системные=0,
    //                      ТолькоЧтение=1,Архивные=1)
    methGetListFiles_Mask:
      begin
        // Скрытые файлы. По умолчанию = 0 (не учитывать)
        if lParamNum=2 then pvarParamDefValue := 0;
        // Системные файлы. По умолчанию = 0 (не учитывать)
        if lParamNum=3 then pvarParamDefValue := 0;
        // ТолькоДляЧтения. По умолчанию = 1
        if lParamNum=4 then pvarParamDefValue := 1;
        // Архивные. По умолчанию = 1
        if lParamNum=5 then pvarParamDefValue := 1;
      end;
    //-----------------------------
    else
      begin
        GetParamDefValue := S_FALSE;
        Exit;
      end;
  end;
  GetParamDefValue := S_OK;
end;
```

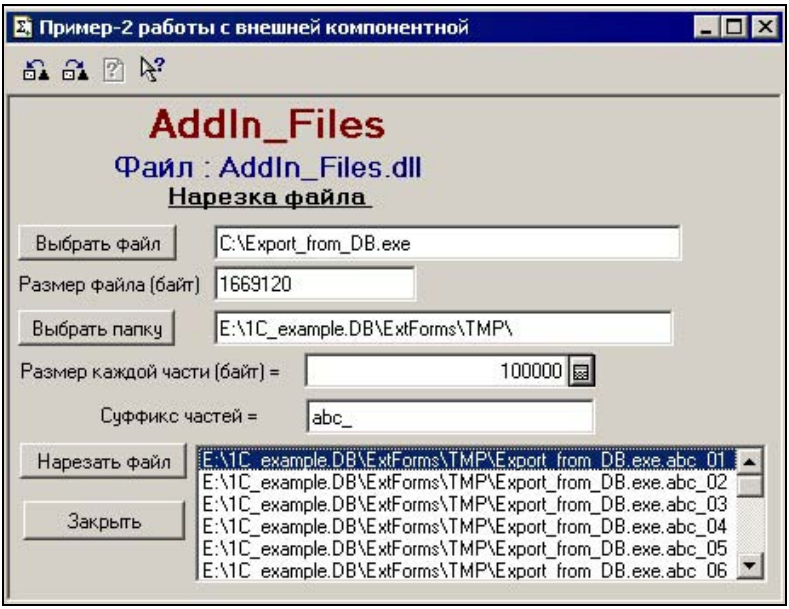



**Рис. 5.22.** Результат тестирования функции `РазрезатьФайл` внешней компоненты `AddIn_Files` (ВнешняяФорма-2_для_AddIn_Files.ERT)

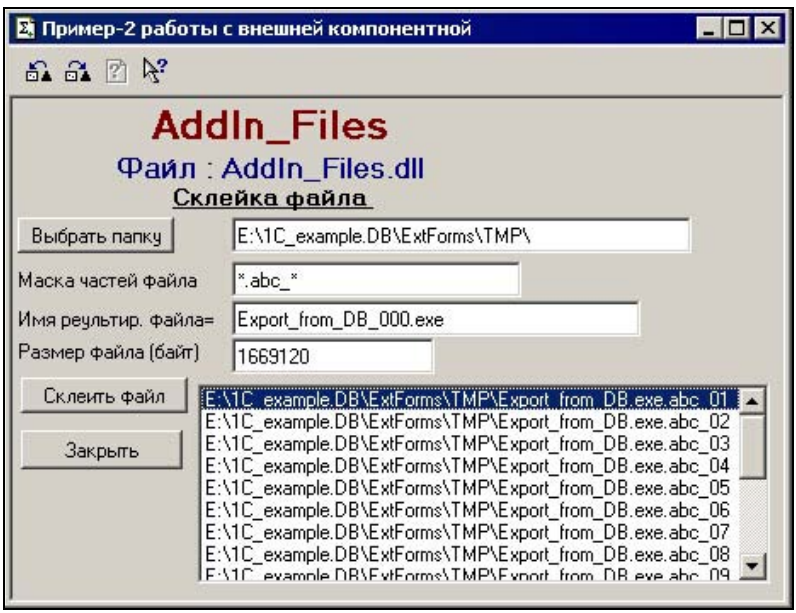



**Рис. 5.23.** Результат тестирования функции `СклеитьФайл` внешней компоненты `AddIn_Files` (ВнешняяФорма-3_для_AddIn_Files.ERT)

На рис. 5.22 представлен результат тестирования функции внешней компоненты `РазрезатьФайл` (обработка ВнешняяФорма-2_для_AddIn_Files.ERT).

На рис. 5.23 представлен результат тестирования функции внешней компоненты `СклеитьФайл` (обработка ВнешняяФорма-3_для_AddIn_Files.ERT).

Внешние компоненты (версии 2.0 и выше) могут создавать свои окна в среде программы 1С:Предприятие с помощью методов интерфейса `IExtWndsSupport` (листинг 5.51).

**Листинг 5.51. Объявление интерфейса `IExtWndsSupport` (модуль AddInLib.PAS)**

```
IExtWndsSupport = interface(IUnknown)
  ['{EFE19EA0-09E4-11D2-A601-008048DA00DE}']
  function GetAppMainFrame(var hwnd: HWND): HResult; stdcall;
  function GetAppMDIFrame(var hwnd: HWND): HResult; stdcall;
  function CreateAddInWindow(const bstrProgID: WideString;
  const bstrWindowName: WideString;
  dwStyles: Integer; dwExStyles: Integer; var rctl: RECT;
  Flags: Integer; var pHwnd: HWND; var pDisp: IDispatch): HResult;
    stdcall;
end;
```

Поддерживаются три типа окон: диалоговые окна (модальные и немодальные) и формы.

*Диалоговые окна* создаются внешней компонентой. *Формы* создаются с помощью метода `CreateAddInWindow` интерфейса `IExtWndsSupport` (листинг 5.52).

> ***Примечание***
>
> Подробную информацию по этому вопросу можно найти в специальной литературе, посвященной вопросам создания внешних компонент для 1С:Предприятие и технологии создания Active Documents.

В рамках этой главы рассмотрим процесс создания форм (из внешней компоненты) на простом примере. Как и в предыдущем случае, за основу возьмем шаблон (sample2), разработанный фирмой 1С.

> ***Примечание***
>
> На сопроводительном компакт-диске книги исходные тексты этого примера находятся в папке \Part_1\Chapter_05\Topic_5_2\AddIn_Example02_01.PAS.

Новую компоненту назовем `AddIn_Example02_01`. Ее функциональность представлена в табл. 5.7.

***Таблица 5.7.*** *Процедуры и функции внешней компоненты* `AddIn_Example02_01`

| **Метод** | **Назначение** |
|---|---|
| `Показать` (или `Show`) | Создать форму и визуализировать ее |
| `ПоказатьСписокСправочников` (или `ShowListRefs`) | Создать форму и заполнить ее списком идентификаторов справочников |
| `ПоказатьСписокДокументов` (или `ShowListDocs`) | Создать форму и заполнить ее списком идентификаторов документов |
| `ПоказатьСписокРегистров` (или `ShowListRegs`) | Создать форму и заполнить ее списком идентификаторов регистров |

Первоначально в рамках шаблона sample2 был реализован (как пример) просмотрщик видеороликов с использованием Delphi-компонента `TMediaPlayer`. Поскольку внешняя компонента в нашем примере ориентирована на другие задачи, пришлось предварительно "вычистить" шаблон.

Прежде всего, нужно изменить содержание файла AddInStr.RC следующим образом (листинг 5.52).

**Листинг 5.52. Содержание файла AddInStr.RC**

```
STRINGTABLE DISCARDABLE
BEGIN
   100      "AddIn_Example02_01"
END
```

Здесь видно, что в строке с `ID = 100` указана вторая часть свойства `ProgID` внешней компоненты.

Теперь нужно с помощью утилиты brc32.EXE, входящей в состав Delphi, скомпилировать файл AddInStr.RC в файл ресурсов AddInStr.RES:

```
brc32.exe AddInStr.rc
```

Далее нужно подключить файл ресурсов к файлу проекта (AddIn_Example02_01.DPR) следующим образом (листинг 5.53).

**Листинг 5.53. Текст файла проекта AddIn_Example02_01.DPR**

```
library AddIn_Example02_01;
uses
  ComServ,
  AddIn_TLB in 'AddIn_TLB.pas',
  AddInDoc in 'AddInDoc.pas'  {AddInDoc: TActiveForm}
                              {AddInDoc: CoClass},
  AddInObj in 'AddInObj.pas',
  AddInLib in 'AddInLib.pas';

{$E dll}

exports
  DllGetClassObject,
  DllCanUnloadNow,
  DllRegisterServer,
  DllUnregisterServer;

{$R *.TLB}

{$R *.RES}
{$R AddInStr.RES}

begin
end.
```

Теперь в форме `ActiveForm` (`TAddInDoc`) шаблона нужно разместить следующие Delphi-компоненты: `Panel1:TPanel`, `Panel2:TPanel`, `ListBox1:TListBox`, `SpeedButton1:TSpeedButton` (рис. 5.24).

**Рис. 5.24.** Форма окна внешней компоненты

Создадим метод, который обеспечит дополнение строк в свойство `Items` Delphi-компонента `ListBox1`. Для этого нужно открыть диалоговое окно редактрования библиотеки типов и внести новый метод `AddLine`, как показано на рис. 5.25.

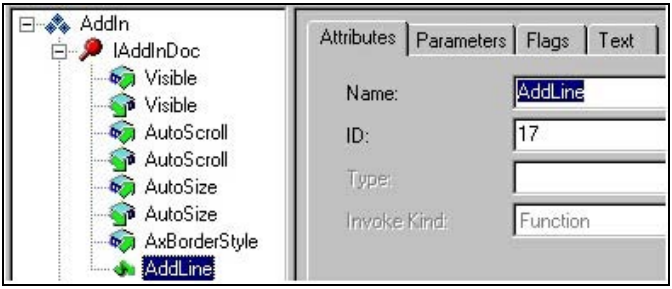



**Рис. 5.25.** Диалоговое окно библиотеки типов

У метода будет один входной параметр (строка, которую нужно добавить в свойство `Items` Delphi-компонента `ListBox1`), рис. 5.26.

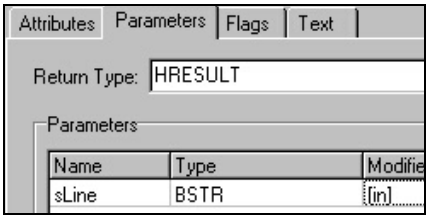



**Рис. 5.26.** Входной параметр метода `AddLine`

Реализация этого метода проста (листинг 5.54).

**Листинг 5.54. Реализация метода `AddLine` в модуле AddInDoc.PAS**

```
procedure TAddInDoc.AddLine(const sLine: WideString);
begin
  ListBox1.Items.ADD(Trim(sLine));
end;
```

Теперь напишем код, реализующий функциональность "нашей" внешней компоненты в методе `CallAsProc` объекта `TAddInObj` (листинг 5.55).

**Листинг 5.55. Реализация функциональности внешней компоненты в методе `CallAsProc`**

```
function TAddInObj.CallAsProc(lMethodNum: Integer;
                              var paParams: PSafeArray): HResult; stdcall;
var
  wnd: HWND;
  cntr: IDispatch;
  rct: RECT;
  form: TAddInDoc;
  obj,V: Variant;
  Sx : string;
  Count,i : integer;
  id : string;
begin
  case TMethods(lMethodNum) of
    //-----------------------------------------------
    // Показать().
    // Здесь просто создается и визуализируется
    // пустое окно ВК (внешней компоненты)
    methShow:
      begin
        form := TAddInDoc.Create(nil);
        rct.left := 10;
        rct.top := 10;
        rct.right := form.Width;
        rct.bottom := form.Height;
        pExtWndsSupport.CreateAddInWindow('AddIn.AddInDoc',
                         'Информационное окно ВК', 0, 0, rct, 0, wnd,
cntr);
        obj := cntr;
        cntr._Release();
      end;
    //-----------------------------------------------
    //-----------------------------------------------
    // ПоказатьСписокСправочников().
    // Здесь создается окно ВК, в котором
    // отображены идентификаторы всех справочников
    // данной конфигурации 1С:Предприятие
```

```
methShowListRefs:
  begin
    form := TAddInDoc.Create(nil);
    rct.left := 10;
    rct.top := 10;
    rct.right := form.Width;
    rct.bottom := form.Height;
    pExtWndsSupport.CreateAddInWindow('AddIn.AddInDoc',
                     'Список справочников', 0, 0, rct, 0, wnd, cntr);
    obj := cntr;
    //----------------------------------------
    // Считывание идентификаторов справочников
    // из метаданных с помощью метода EvalExpr()
    // и их отображение в окне ВК с помощью метода AddLine()
    V:=pConn;
    V := V.AppDispatch;
    IDispatch(V)._AddRef;
    Count:=1*V.EvalExpr('Метаданные.Справочник()');
    for i:=1 to Count do
     begin
       Sx:='Метаданные.Справочник('+IntToStr(i)+')';
       id:=Trim(V.EvalExpr(Sx).Идентификатор);
       obj.AddLine(id);
    end;
    //----------------------------------------
    cntr._Release();
  end;
//-----------------------------------------------
//-----------------------------------------------
// ПоказатьСписокДокументов().
// Здесь создается окно ВК, в котором
// отображены идентификаторы всех документов
// данной конфигурации 1С:Предприятие
methShowListDocs:
  begin
    form := TAddInDoc.Create(nil);
    rct.left := 10;
    rct.top := 10;
```

```
    rct.right := form.Width;
    rct.bottom := form.Height;
    pExtWndsSupport.CreateAddInWindow('AddIn.AddInDoc',
                    'Список Документов', 0, 0, rct, 0, wnd, cntr);
    obj := cntr;
    //----------------------------------------
    // Считывание идентификаторов документов
    // из метаданных с помощью метода EvalExpr()
    // и их отображение в окне ВК с помощью метода AddLine()
    V:=pConn;
    V := V.AppDispatch;
    IDispatch(V)._AddRef;
    Count:=1*V.EvalExpr('Метаданные.Документ()');
    for i:=1 to Count do
     begin
       Sx:='Метаданные.Документ('+IntToStr(i)+')';
       id:=Trim(V.EvalExpr(Sx).Идентификатор);
       obj.AddLine(id);
    end;
    //----------------------------------------
    cntr._Release();
  end;
//-----------------------------------------------
//-----------------------------------------------
// ПоказатьСписокРегистров().
// Здесь создается окно ВК, в котором
// отображены идентификаторы всех регистров
// данной конфигурации 1С:Предприятие
methShowListRegs:
  begin
    form := TAddInDoc.Create(nil);
    rct.left := 10;
    rct.top := 10;
    rct.right := form.Width;
    rct.bottom := form.Height;
    pExtWndsSupport.CreateAddInWindow('AddIn.AddInDoc',
                    'Список регистров', 0, 0, rct, 0, wnd, cntr);
    obj := cntr;
```

```
        //-------------------------------------
        // Считывание идентификаторов регистров
        // из метаданных с помощью метода EvalExpr()
        // и их отображение в окне ВК с помощью метода AddLine()
        V:=pConn;
        V := V.AppDispatch;
        IDispatch(V)._AddRef;
        Count:=1*V.EvalExpr('Метаданные.Регистр()');
        for i:=1 to Count do
         begin
           Sx:='Метаданные.Регистр('+IntToStr(i)+')';
           id:=Trim(V.EvalExpr(Sx).Идентификатор);
           obj.AddLine(id);
        end;
        //-------------------------------------
        cntr._Release();
      end;
    //-----------------------------------------------
    else
      begin
         CallAsProc := S_FALSE;
         Exit;
      end;
  end;
  CallAsProc := S_OK;
end;
```

В принципе, здесь используются приемы, рассмотренные ранее в этой и предыдущих главах. Единственное, что следует отметить: при создании нового окна (активного документа) метод `CreateAddInWindow` интерфейса `IExtWndsSupport` возвращает указатель на интерфейс `IDispatch (cntr)` созданного активного документа, что позволяет управлять им из внешней компоненты, обращаясь к его методам и свойствам. Назначение параметров метода `CreateAddInWindow`:

```
function CreateAddInWindow
(const bstrProgID: WideString;     // ProgID Активного Документа
 const bstrWindowName: WideString; // Заголовок окна
 dwStyles: Integer;     // Стиль окна (0 по умолчанию)
```

```
 dwExStyles: Integer;   // Дополнительный стиль окна (0 по умолчанию)
 var rctl: RECT;       // Размеры и положение окна
 Flags: Integer;       // Резерв (должен быть равен 0)
 var pHwnd: HWND;      // Указатель на дескриптор созданного окна
 var pDisp: IDispatch // Указатель на IDispatch
                       // созданного "Активного Документа"
): HResult; stdcall;
```

На рис. 5.27 представлена форма внешней обработки ВнешняяФорма_для_AddIn_Example02_01.ERT, предназначенная для тестирования внешней компоненты `AddIn_Example02_01`.

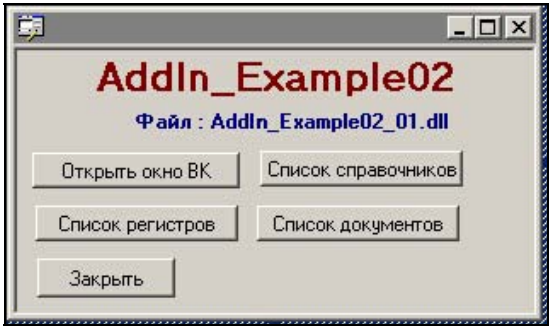



**Рис. 5.27.** Форма внешней обработки для тестирования внешней компоненты `AddIn_Example02_01`

Текст программного модуля внешней обработки представлен в листинге 5.56.

**Листинг 5.56. Программный модуль внешней обработки**

```
Перем ВК Экспорт;

Процедура ОткрытьИнформОкноВК()
  ВК.Показать();
КонецПроцедуры

Процедура ПоказатьСписокСправочников()
  ВК.ПоказатьСписокСправочников();
КонецПроцедуры

Процедура ПоказатьСписокДокументов()
```

```
  ВК.ПоказатьСписокДокументов();
КонецПроцедуры

Процедура ПоказатьСписокРегистров()
  ВК.ПоказатьСписокРегистров();
КонецПроцедуры

Процедура ПриОткрытии()
  ЗагрузитьВнешнююКомпоненту("AddIn_Example02_01.dll");
  ВК = СоздатьОбъект("AddIn.AddIn_Example02");
КонецПроцедуры
```

На рис. 5.28 представлен результат тестирования внешней компоненты `AddIn_Example02_01`.

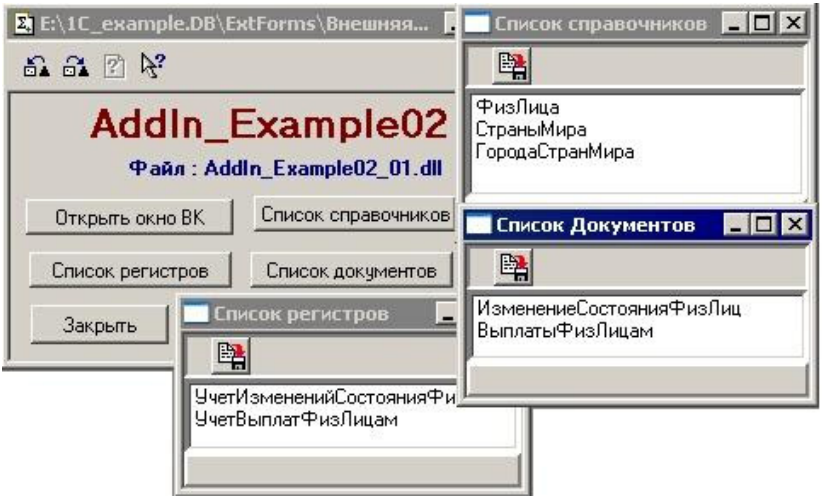



**Рис. 5.28.** Результат тестирования внешней компоненты `AddIn_Example02_01`

Для одной внешней компоненты можно создать несколько форм типа `TActiveForm`. Чтобы рассмотреть этот вопрос на конкретном примере, расширим функциональность компоненты `AddIn_Example02_01`. Добавим в нее что-то вроде простой записной книжки. Записная книжка представляет собой набор пронумерованных текстовых файлов (на каждую букву алфавита — отдельный текстовый файл). Вызов записной книжки из программы

1С:Предприятие будет производиться с помощью процедуры `ПоказатьМоиЗаписи` или `ShowMyNotes`. В качестве входного параметра процедура принимает имя папки, где будут размещаться файлы записной книжки. Таким образом, задавая имя папки, мы, по сути, выбираем определенную записную книжку.

Новую компоненту назовем `AddIn_Example02_02`.

> ***Примечание***
>
> На сопроводительном компакт-диске книги исходные тексты этого примера находятся в папке \Part_1\Chapter_05\Topic_5_2\AddIn_Example02_02.PAS.

Чтобы создать новую форму, нужно выбрать пункт меню **File > New**, в открывшемся диалоговом окне перейти на вкладку **ActiveX** и выбрать значок **ActiveForm** (см. рис. 5.1). На экране появится диалоговое окно настройки свойств формы (рис. 5.29).

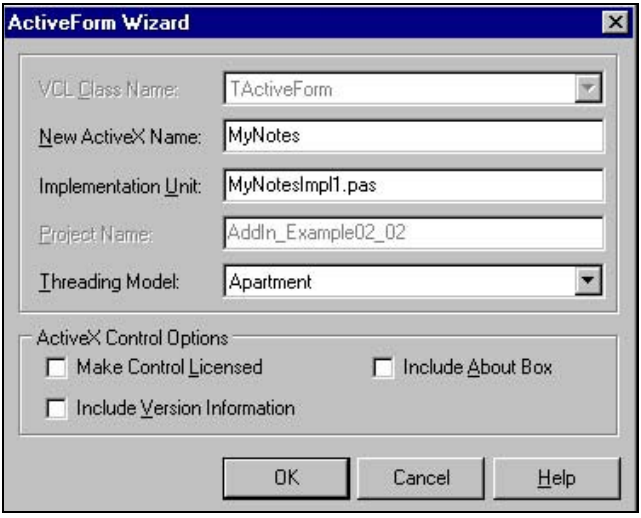



**Рис. 5.29.** Новая форма типа `ActiveForm` внешней компоненты `AddIn_Example02_02`

Создав форму, нужно будет открыть файл проекта AddIn_Example02_02.DPR и заменить строку `{$E ocx}` строкой `{$E dll}`.

На рис. 5.30 представлена форма записной книжки.

В листинге 5.57 представлена реализация функциональности процедуры `ПоказатьМоиЗаписи` в методе `SaveAsProc` объекта `TAddInObj`.

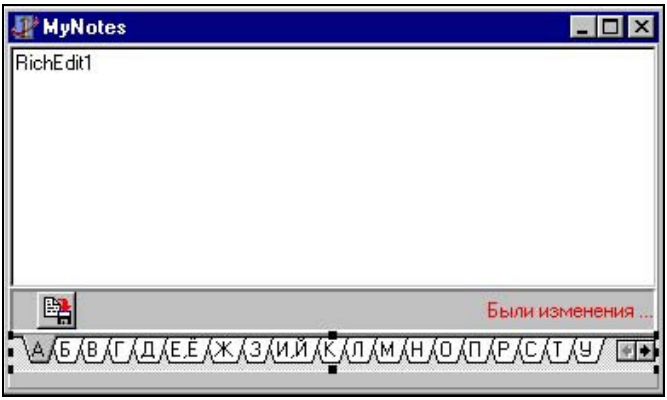



**Рис. 5.30.** Форма `TMyNotes` внешней компоненты `AddIn_Example02_02`

**Листинг 5.57. Реализация процедуры `ПоказатьМоиЗаписи` в методе `CallAsProc` объекта `TAddInObj`**

```
// ПоказатьМоиЗаписи(DirName_for_Notes)
methShowMyNotes:
  begin
    // В первом входном параметре
    // получаем имя папки, где хранятся файлы
    // записной книжки
    Sx:=Trim((GetNParam(paParams,0)));
    if length(Sx)>0 then begin
       form2 := TMyNotes.Create(nil);
       rct.left := 10;
       rct.top := 10;
       rct.right := form2.Width;
       rct.bottom := form2.Height;
       pExtWndsSupport.CreateAddInWindow('AddIn.MyNotes',
                          'Моя записная книжка', 0, 0, rct, 0,
                          wnd, cntr);
       obj := cntr;
       // Запоминаем имя папки
       obj.SetDirName(Sx);
       // Загружаем текстовый файл для первой закладки
       // (буква А)
```

```
        obj.LoadThisPage(0);
        //
        cntr._Release();
    end;
  end;
```

На рис. 5.31 представлена форма внешней обработки ВнешняяФорма_для_AddIn_Example02_02.ERT системы 1С:Предприятие для тестирования внешней компоненты AddIn_Example02_02.

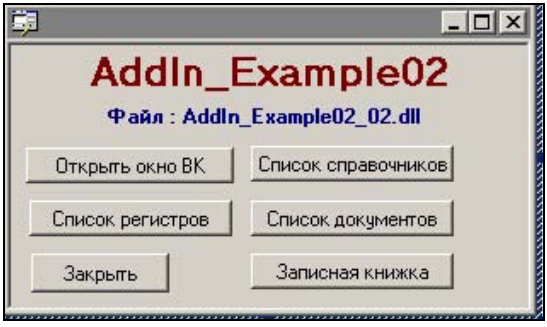



**Рис. 5.31.** Форма внешней обработки для тестирования внешней компоненты AddIn_Example02_02

В листинге 5.58 представлен текст программного модуля обработки.

**Листинг 5.58. Программный модуль обработки ВнешняяФорма_для_AddIn_Example02_02.ERT**

```
Перем ВК Экспорт;

Процедура ОткрытьИнформОкноВК()
  ВК.Показать();
КонецПроцедуры

Процедура ПоказатьСписокСправочников()
  ВК.ПоказатьСписокСправочников();
КонецПроцедуры

Процедура ПоказатьСписокДокументов()
  ВК.ПоказатьСписокДокументов();
```

```
КонецПроцедуры

Процедура ПоказатьСписокРегистров()
  ВК.ПоказатьСписокРегистров();
КонецПроцедуры

Процедура ЗаписнаяКнижка()
  ВК.ПоказатьМоиЗаписи(каталогИБ()+"ExtForms\"+"TMP\");
КонецПроцедуры

Процедура ПриОткрытии()
  ЗагрузитьВнешнююКомпоненту("AddIn_Example02_02.dll");
  ВК = СоздатьОбъект("AddIn.AddIn_Example02");
КонецПроцедуры
```

На рис. 5.32 представлен результат тестирования внешней компоненты `AddIn_Example02_02`.

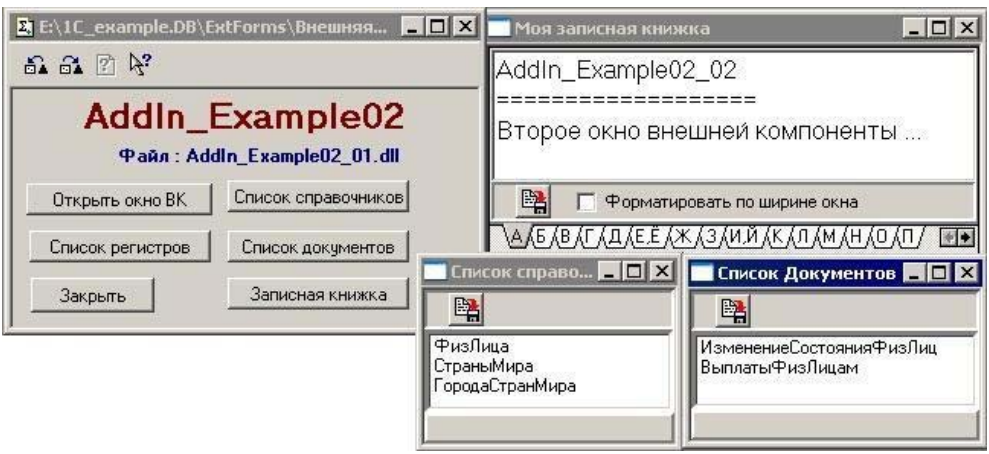



**Рис. 5.32.** Результат тестирования внешней компоненты `AddIn_Example02_02`

### *Примечание*

Напомню, что приведенные примеры исходных текстов — это упрощенная иллюстрация к рассматриваемым вопросам. Назначение этих примеров — дать представление о возможном подходе при решении задач информационного обмена с системой 1С:Предприятие. При работе с реальными базами 1С может потребоваться их дальнейшая доработка.

# 5.3. Внепроцессный сервер автоматизации

Внепроцессный сервер автоматизации выполняется в собственном адресном пространстве (отличном от адресного пространства контроллера). Отсюда следует, что для управления сервером клиент должен обращаться к свойствам и методам объектов, находящихся в ином адресном простанстве. Для этого используется технология LRPC (Local Remote Procedure Calls). Как правило, большинство внепроцессных серверов автоматизации являются полноценными автономными приложениями. С другой стороны, приложение легко можно превратить во внепроцессный сервер автоматизации.

В этом разделе мы на конкретном примере рассмотрим, каким образом можно использовать внепроцессный сервер автоматизации для расширения функциональности системы 1С:Предприятие.

Создадим новый Delphi-проект (рис. 5.33).

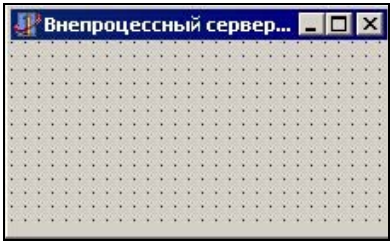



**Рис. 5.33.** Пустая форма Delphi-проекта

Сохраним проект под именем oleSrv1C.DPR и, не внося больше изменений в проект, выберем пункт меню **File > New**. В открывшемся диалоговом окне на вкладке **ActiveX** нужно выделить значок (пиктограмму) **Automation Object** (рис. 5.34) и нажать кнопку **OK**.

На экране появится диалоговое окно настройки свойств объекта **Automation Object** (рис. 5.35).

В *разд. 5.2* это окно было рассмотрено достаточно подробно, поэтому здесь углубляться не будем. Отметим только, что свойство `ProgID` сервера автоматизации будет установлено так:

`oleSrv1C.`**`Tools`**

Закрыв окно нажатием кнопки **OK**, нужно сохранить и скомпилировать проект oleSrv1C.DPR. Полученное приложение oleSrv1C.EXE и есть внепроцессный сервер автоматизации.

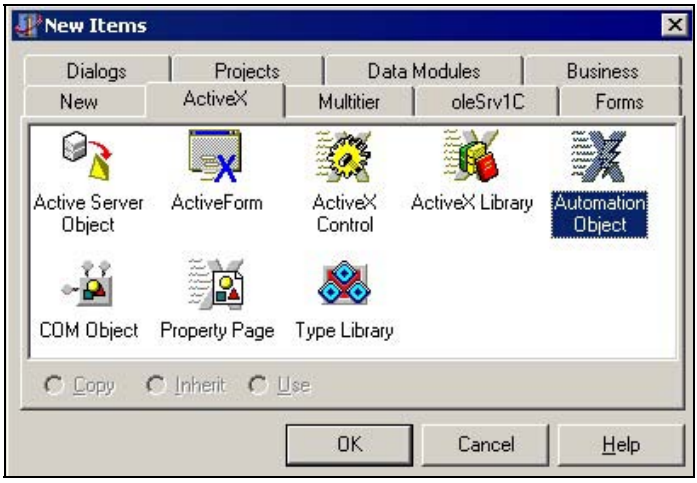



**Рис. 5.34**. Выбор значка **Automation Object**

**Рис. 5.35.** Диалоговое окно настройки свойств **Automation Object**

***Примечание***

На сопроводительном компакт-диске книги исходные тексты этого примера находятся в папке \Part_1\Chapter_05\Topic_5_3\oleSrv1C_01.PAS.

Для того чтобы зарегистрировать сервер в операционной системе (в реестре), нужно просто запустить приложение oleSrv1C.EXE — и все. Очевидно, что создать внепроцессный сервер автоматизации существенно проще, чем внутрипроцессный *(см. разд. 5.2)*.

Теперь наделим наш сервер автоматизации каким-нибудь свойством, например, позволяющим делать невидимой форму главного окна приложения (сервера).

Для этого откроем диалоговое окно библиотеки типов и введем свойство `Visible` (рис. 5.36).

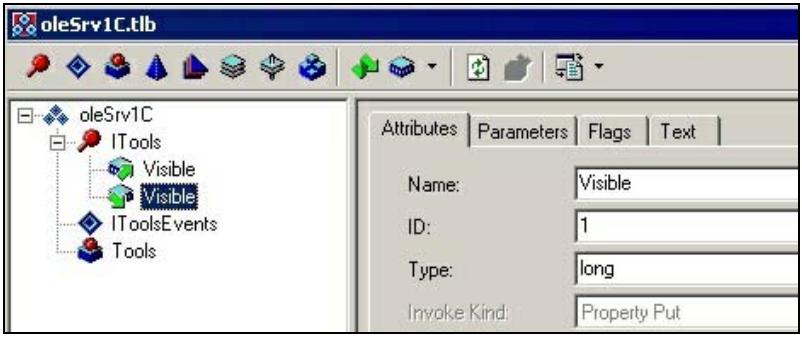



**Рис. 5.36.** Диалоговое окно библиотеки типов

В листинге 5.59 представлена реализация свойства `Visible` сервера автоматизации.

**Листинг 5.59. Реализация свойства `Visible` сервера автоматизации oleSrv1C.EXE**

```
function TTools.Get_Visible: Integer;
// Считывание свойства Visible
begin
  Result:=0;
  if Form1.Visible then Result:=1;
end;

procedure TTools.Set_Visible(Value: Integer);
// Запись свойства Visible
begin
  if Value>0 then begin
     Form1.Visible:=TRUE;
  end
  else begin
     Form1.Visible:=FALSE;
  end;
end;
```

Здесь все очевидно.

Теперь нужно скомпилировать проект и запустить приложение oleSrv1C.EXE для того, чтобы оно само себя зарегистрировало как сервер OLE Automation. Это достаточно сделать один раз. В дальнейшем, при обращении к серверу какого-либо другого приложения (контроллера автоматизации) операционная система сама найдет и запустит oleSrv1C.EXE.

Чтобы протестировать сервер автоматизации, откроем Конфигуратор 1С:Предприятие и создадим внешнюю обработку (рис. 5.37).

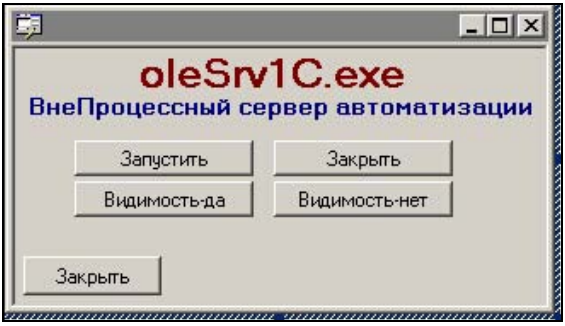



**Рис. 5.37.** Внешняя обработка для тестирования oleSrv1C.EXE

Имя файла этой обработки — ВнешняяФорма-1_для_oleSrv1C_exe.ERT. Текст программного модуля обработки представлен в листинге 5.60.

**Листинг 5.60. Программный модуль обработки ВнешняяФорма-1_для_oleSrv1C_exe.ERT**

```
Перем oleSrv1C;

Процедура Запустить_oleSrv1C()
  oleSrv1C = СоздатьОбъект("oleSrv1C.Tools");
КонецПроцедуры

Процедура Закрыть_oleSrv1C()
  oleSrv1C = "";
КонецПроцедуры

Процедура ВидимостьДА()
  oleSrv1C.Visible = 1;
```

```
КонецПроцедуры

Процедура ВидимостьНЕТ()
  oleSrv1C.Visible = 0;
КонецПроцедуры
```

Результат тестирования представлен на рис. 5.38.

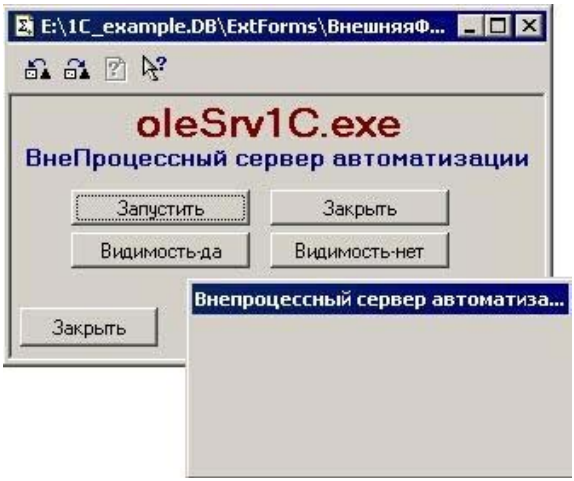



**Рис. 5.38.** Результат тестирования сервера oleSrv1C.EXE

По нажатию кнопки **Запустить** на экране компьютера появится окно сервера автоматизации. По нажатию кнопки **Видимость-нет** окно сервера станет невидимым (при этом сам сервер остается в памяти). По нажатию кнопки **Видимость-да** окно сервера вновь появится на экране и т. д. Чтобы выгрузить сервер, нужно просто закрыть форму или "обнулить" переменную, где хранится ссылка на сервер (в данном случае переменная `oleSrv1C` (см. листинг 5.60)).

Расширим функциональность сервера. Для этого вновь откроем диалоговое окно библиотеки типов и введем новый метод `IsDate` (рис. 5.39).

Этот метод получает во входном параметре `sDate` строку, которая представляет собой дату в допустимом формате (например, *ДД.ММ.ГГГГ* или *ДД/ММ/ГГГГ*). Метод должен проанализировать строку и вернуть значение 1, если дата корректна, или 0 в противном случае. Реализация этого метода приведена в листинге 5.61.

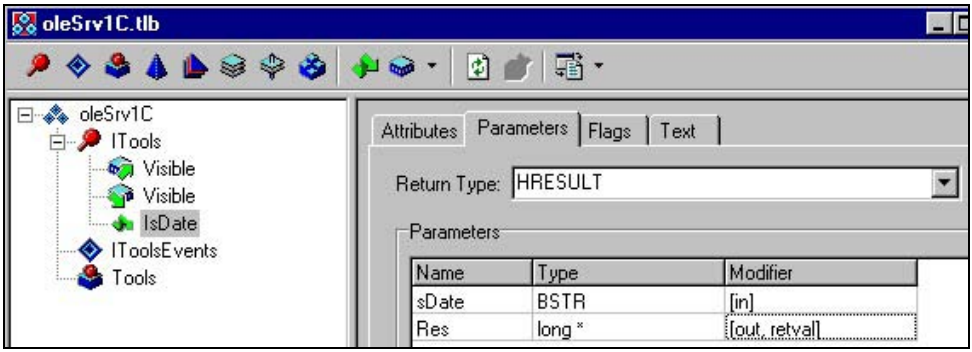



**Рис. 5.39.** Расширение функциональности сервера автоматизации: новый метод `IsDate`

**Листинг 5.61. Реализация метода `IsDate` сервера автоматизации oleSrv1C.EXE**

```
function TTools.IsDate(const sDate: WideString): Integer;
// Проверка строки sDate на корректность, если это дата
begin
  Result:=0;
  if sp_IsDate(Trim(sDate)) then Result:=1;
end;
```

Текст функции `sp_IsDate` здесь не приводится из экономии места (полный текст функции можно посмотреть в файле os1c_00.PAS данного примера).

Следует отметить, что проверка даты на корректность особенно актуальна, когда производится считывание информации из программы 1С:Предприятие через механизм OLE Automation. В случае "пустого" значения даты это становится существенной проблемой при "стандартном" подходе.

Проблему можно решить, обеспечив соединение одной программы 1С:Предприятие (как клиента автоматизации) с другой программой 1С:Предприятие (как с сервером автоматизации) с помощью сервера oleSrv1C.EXE.

То есть сервер oleSrv1C.EXE будет выполнять роль посредника. В этом случае, как ни странно, метод `IsDate` работает корректно (то есть воспринимает "пустую" дату как строку с ошибочной датой).

Попробуем применить это на практике. Создадим новый метод `ConnectTo1C` (рис. 5.40).

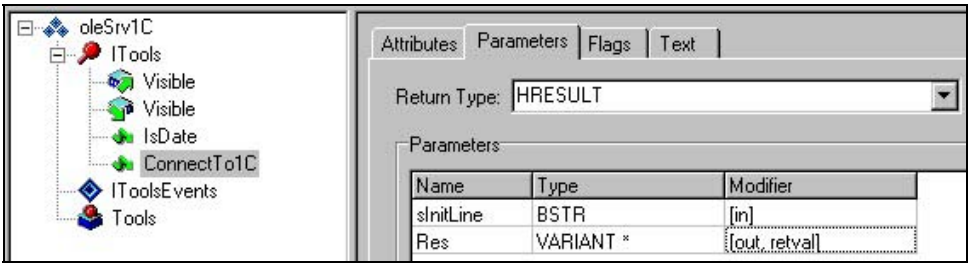



**Рис. 5.40.** Расширение функциональности сервера автоматизации: новый метод `ConnectTo1C`

Его назначение — запуск программы 1С:Предприятие как сервера OLE Automation и возврат ссылки на него в вызывающий программный модуль. Реализация этого метода приведена в листинге 5.62.

**Листинг 5.62. Метод `ConnectTo1C` сервера автоматизации oleSrv1C.EXE**

```
function TTools.ConnectTo1C(const sInitLine: WideString): OleVariant;
// Запуск программы 1С:Предприятие в режиме сервера OLE Automation:
// sInitLine – строка инициализации программы 1С:Предприятие
begin
  Result:=ConnectTo1C_via(Trim(sInitLine));
end;
```

В листинге 5.63 приведен текст функции `ConnectTo1C_via`.

**Листинг 5.63. Функция `ConnectTo1C_via`**

```
function ConnectTo1C_via(InitStr : string) : OleVariant;
// Запуск программы 1С:Предприятие в режиме сервера OLE Automation:
// InitLine – строка инициализации программы 1С:Предприятие
Var
  ResOpen : integer;
begin
  Result:=UnAssigned;
  InitStr:=Trim(InitStr);
  Result := CreateOleObject('V77.Application');
  ResOpen:=Result.Initialize(Result.RMTrade, InitStr, '');
  if ResOpen>=0 then begin
```

```
    Result:=UnAssigned;
  end;
end;
```

Сохраним и скомпилируем наш Delphi-проект.

Чтобы протестировать функцию `IsDate`, запустим программу 1С:Предприятие, откроем кофигурацию 1С_example.DB2 и очистим значение реквизита `ДатаРождения` для одного из элементов справочника `ФизЛица`, например, для элемента справочника с кодом 5 (рис. 5.41).

| Код | Наименование | Дата рож... |
|---|---|---|
| 1 | Иванов Иван Иванович | 01.01.71 |
| 2 | Петров Петр Петрович | 01.01.72 |
| 3 | Сидоров Сидор Сидорович | 01.01.73 |
| 4 | Федоров Федор Федорович | 01.01.74 |
| 5 | Аксакалов Аксакал Аксакалович | . . |
| 6 | Криогенова Фекла Маринадовна | 15.03.65 |

**Рис. 5.41.** Содержание справочника `ФизЛица` в конфигурации 1С_example.DB2

Закроем программу 1С:Предприятие.

Теперь запустим Конфигуратор 1С:Предприятие, откроем конфигурацию 1С_example.DB и создадим внешнюю обработку ВнешняяФорма-2_для_oleSrv1C_exe.ERT (рис. 5.42).

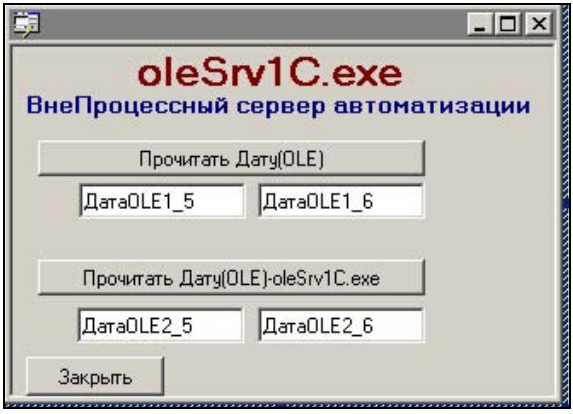



**Рис. 5.42.** Форма внешней обработки для тестирования функции `IsDate` сервера автоматизации

Здесь требуются некоторые пояснения. Попробуем прочитать значение реквизита `ДатаРождения` элемента справочника `ФизЛица` с кодом 5 двумя способами.

- *Способ 1 ("стандартный"):* из программного модуля 1С:Предприятие вызывается также программа 1С:Предприятие в качестве сервера OLE Automation следующим образом:

```
Ole1C = СоздатьОбъект("V77.Application");
Ole1C.Initialize(Ole1C.RMTrade, "", "");
```

- *Способ 2:* в программном модуле 1С:Предприятие вызывается сначала сервер автоматизации oleSrv1C.EXE, а затем с помощью его метода `ConnectTo1C` вызывается программа 1С:Предприятие в качестве сервера OLE Automation следующим образом:

```
oleSrv1C = СоздатьОбъект("oleSrv1C.Tools");
oleSrv1C.Visible = 0;
Ole1C = oleSrv1C.ConnectTo1C("");
```

Для чистоты эксперимента попутно прочитаем значения реквизита `ДатаРождения` элемента справочника `ФизЛица` с кодом 6 обоими способами.

Назначение элементов управления в форме обработки приведено в табл. 5.8.

***Таблица 5.8.*** *Назначение элементов управления формы обработки ВнешняяФорма-2_для_oleSrv1C_exe.ERT*

| Элемент управления | Тип | Назначение |
|---|---|---|
| `ДатаOLE1_5` | `Дата` | Значение реквизита `ДатаРождения` элемента справочника `ФизЛица` с кодом **5** при считывании через механизм OLE (способ 1) |
| `ДатаOLE1_6` | `Дата` | Значение реквизита `ДатаРождения` элемента справочника `ФизЛица` с кодом **6** при считывании через механизм OLE (способ 1) |
| `ДатаOLE2_5` | `Дата` | Значение реквизита `ДатаРождения` элемента справочника `ФизЛица` с кодом **5** при считывании через механизм OLE (способ 2) |
| `ДатаOLE2_6` | `Дата` | Значение реквизита `ДатаРождения` элемента справочника `ФизЛица` с кодом **6** при считывании через механизм OLE (способ 2) |

***Таблица 5.8*** *(окончание)*

| Элемент управления | Тип | Назначение |
|---|---|---|
| Прочитать Дату(OLE) | Кнопка | Соединение с 1С:Предприятие (способ 1). Считывание значений реквизита ДатаРождения элементов справочника ФизЛица с кодами **5** и **6** через механизм OLE |
| Прочитать Дату(OLE)-oleSrv1C.exe | Кнопка | Соединение с 1С:Предприятие (способ 2). Считывание значений реквизита ДатаРождения элементов справочника ФизЛица с кодами **5** и **6** через механизм OLE |

В листинге 5.64 представлен текст программного модуля внешней обработки ВнешняяФорма-2_для_oleSrv1C_exe.ERT.

**Листинг 5.64. Программный модуль внешней обработки ВнешняяФорма-2_для_oleSrv1C_exe.ERT**

```
Перем oleSrv1C;

//==================================
// Считывание информации из 1C_example.DB2 посредством механизма OLE.
// Способ 1: соединение "напрямую"
Процедура ПрочитатьДатуOLE()
  ДатаOLE1_5 = "";
  ДатаOLE1_6 = "";
  Ole1C = СоздатьОбъект("V77.Application");
  Ole1C.Initialize(Ole1C.RMTrade, "", "");
  СпрФизЛица = Ole1C.CreateObject("Справочник.ФизЛица");

  //-----------------------------------
  //Здесь дата рождения пустая
  Если СпрФизЛица.НайтиПоКоду(5,0)>0 Тогда
    Sx = СпрФизЛица.ДатаРождения;
    ДатаOLE1_5 = ДАТА(Sx);
  КонецЕсли;
  //-----------------------------------

  //-----------------------------------
```

```
  //Здесь дата рождения = 15.03.65
  Если СпрФизЛица.НайтиПоКоду(6,0)>0 Тогда
    Sx = СпрФизЛица.ДатаРождения;
    ДатаOLE1_6 = ДАТА(Sx);
  КонецЕсли;
  //-------------------------------------

КонецПроцедуры
//==================================

//==================================
// Считывание информации из 1C_example.DB2 посредством механизма OLE.
// Способ 2: соединение через oleSrv1C.EXE
Процедура ПрочитатьДатуOLE_2()
  ДатаOLE2_5 = "";
  ДатаOLE2_6 = "";
  oleSrv1C = СоздатьОбъект("oleSrv1C.Tools");
  oleSrv1C.Visible = 0;
  Ole1C = oleSrv1C.ConnectTo1C("");
  СпрФизЛица = Ole1C.CreateObject("Справочник.ФизЛица");

  //-------------------------------------
  // Здесь дата рождения пустая
  Если СпрФизЛица.НайтиПоКоду(5,0)>0 Тогда
    Sx = СпрФизЛица.ДатаРождения;
    Если oleSrv1C.IsDate(Sx)>0 Тогда
      ДатаOLE2_5 = ДАТА(Sx);
    Иначе
      Сообщить("Ошибка в дате="+Sx);
    КонецЕсли;
  КонецЕсли;
  //-------------------------------------

  //-------------------------------------
  //Здесь дата рождения = 15.03.65
  Если СпрФизЛица.НайтиПоКоду(6,0)>0 Тогда
    Sx = СпрФизЛица.ДатаРождения;
    Если oleSrv1C.IsDate(Sx)>0 Тогда
```

```
      ДатаOLE2_6 = ДАТА(Sx);
    Иначе
      Сообщить("Ошибка в дате="+Sx);
    КонецЕсли;
  КонецЕсли;
  //-----------------------------------

КонецПроцедуры
//==================================
```

Теперь запустим программу 1С:Предприятие, откроем внешнюю обработку ВнешняяФорма-2_для_oleSrv1C_exe.ERT и протестируем оба способа (естественно, нужно выбрать одну и ту же конфигурацию 1C_example.DB2). Результат тестирования представлен на рис. 5.43.

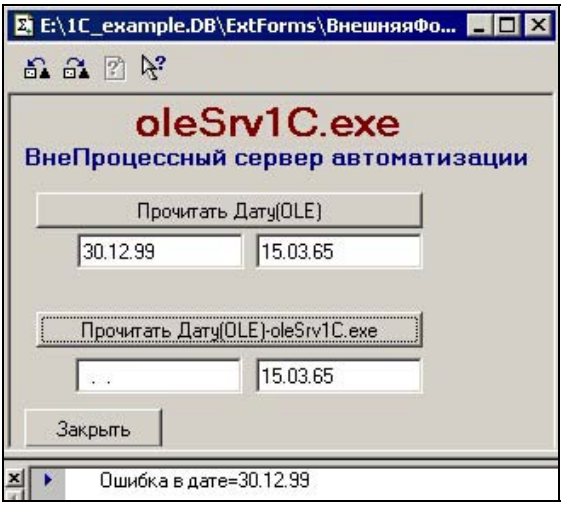



**Рис. 5.43.** Результат тестирования функции `IsDate` сервера oleSrv1C.EXE

Здесь комментарии излишни.

У внешних компонент для системы 1С:Предприятие в виде внутрипроцессных серверов автоматизации *(см. разд. 5.2)* есть свои преимущества. Но один из их существенных недостатков — трудоемкость в разработке и отладке функционально сложных *внутрипроцессных* серверов OLE Automation для программы 1С:Предприятие. Тогда как использование *внепроцессных* серверов автоматизации существенно упрощает и ускоряет решение этой задачи. Рассмотрим этот момент на следующем примере.

Папка \1C_example.DB\ExtForms\ сопроводительного компакт-диска книги содержит демонстрационный EXE-модуль oleExecScript.EXE, представляющий собой внепроцессный сервер автоматизации, функциональность которого кратко описана в табл. 5.9.

***Таблица 5.9.*** *Методы и свойства сервера OLE Automation oleExecScript.EXE*

| **Метод/Свойство** | **Назначение** |
|---|---|
| `Visible` (свойство) | Видимость главного окна приложения: если значение этого свойства больше 0, то окно сервера видимо, в противном случае — нет |
| `StayOnTop` (свойство) | Режим визуализации главного окна приложения: если значение этого свойства больше 0, то окно приложения находится в режиме "всегда сверху", в противном случае — нет |
| `EvalExpr` (метод) | Вычисление выражения. Синтаксис:<br>`EvalExpr(`*`СтрокаВыражения`*`,`*`КодЯзыка`*`);`<br>*`КодЯзыка`*: 0 (по умолчанию) — Pascal. Возвращаемое значение: `VARIANT` |
| `ExecuteBatch` (метод) | Выполнение скрипта (сценария). Синтаксис:<br>`ExecuteBatch(`*`ТекстСкрипта`*`,`*`КодЯзыка`*`,`*`СтрокаОшибки`*`);`<br>*`ТекстСкрипта`*: строка, содержащая текст выполняемого скрипта; *`КодЯзыка`*: 0 (по умолчанию) — Pascal. Возвращаемое значение: `Integer` (больше 0 — успешное выполнение, в противном случае — нет) |
| `ExecuteBatchF` (метод) | Выполнение скрипта из файла. Синтаксис:<br>`ExecuteBatchF(`*`ИмяФайлаСкрипта`*`,`*`КодЯзыка`*`,`<br>*`СтрокаОшибки`*`);`<br>*`КодЯзыка`*: 0 (по умолчанию) — Pascal. Возвращаемое значение: `Integer` (больше 0 — успешное выполнение, в противном случае — нет) |
| `GetVarValue` (метод) | Получение значения переменной выполненного скрипта. Синтаксис:<br>`GetVarValue(`*`ИмяПеременной`*`);`<br>Возвращаемое значение: `VARIANT` |
| `GetVarCount` (метод) | Получение количества переменных выполненного скрипта. Синтаксис:<br>`GetVarCount();` |

***Таблица 5.9*** *(окончание)*

| Метод/Свойство | Назначение |
|---|---|
| `GetVarName` (метод) | Получение имени переменной по ее номеру (начиная с нуля). Синтаксис: `GetVarName(`*НомерПеременной*`);` Возвращаемое значение: `String` |
| `GetVarNamesString` (метод) | Получение имен всех переменных. Синтаксис: `GetVarNamesString();` Возвращаемое значение: `String` |

***Примечание***

Указанная функциональность сервера была реализована с помощью пакета FastScript версии 1.4 (Copyright 1998—2004 by Fast Reports Inc., Author: Alexander Tzyganenko, **http://www.fastreport.ru**), который представляет собой прекрасную библиотеку для выполнения скриптов с поддержкой синтаксиса PascalScript, C++Script, BasicScript, JScript. Этот пакет имеет статус коммерческого программного продукта, поэтому исходные тексты программы oleExecScript.EXE здесь не приводятся.

Запустим Конфигуратор 1С:Предприятие и создадим внешнюю обработку ВнешняяФорма-1_для_oleExecScript.ERT (рис. 5.44).

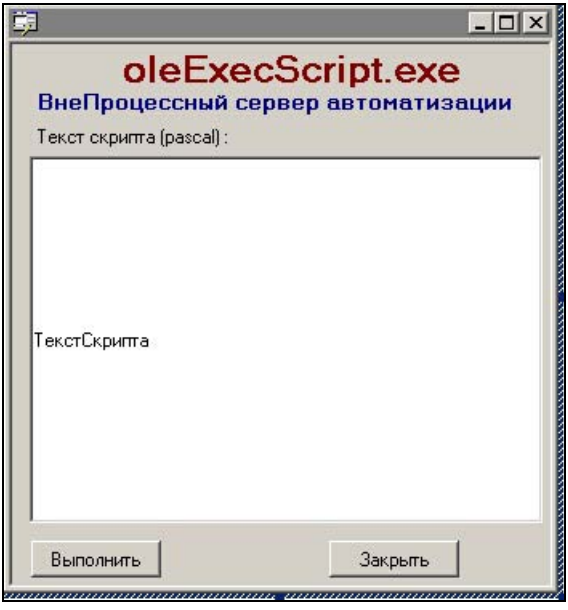



**Рис. 5.44.** Внешняя обработка для тестирования сервера oleExecScript.EXE

В листинге 5.65 приведен текст программного модуля внешней обработки ВнешняяФорма-1_для_oleExecScript.ERT.

**Листинг 5.65. Программный модуль обработки ВнешняяФорма-1_для_oleExecScript.ERT**

```
Перем oleExecScript;

Процедура ВыполнитьСкрипт()

  СтрокаОшибки="";
  //-----------------------------
  // Выполнение скрипта
  Если oleExecScript.ExecuteBatch(ТекстСкрипта,0,СтрокаОшибки)<=0 Тогда
     Предупреждение(СтрокаОшибки);
  КонецЕсли;
  //-----------------------------

КонецПроцедуры
```

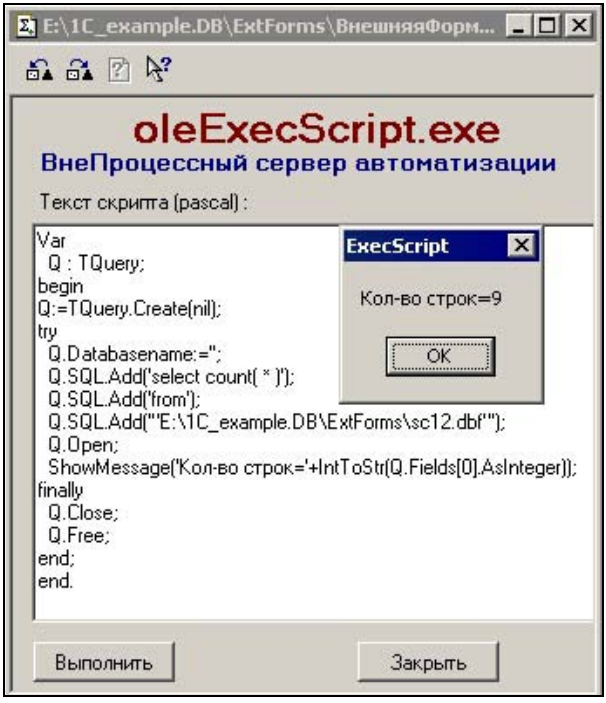



**Рис. 5.45.** Результат тестирования метода `ExecuteBatch` сервера oleExecScript.EXE

Запустим программу 1С:Предприятие, откроем файл созданной обработки, введем текст скрипта и запустим его на выполнение (рис. 5.45).

> ***Примечание***
>
> Для выполнения указанного скрипта требуется, чтобы на компьютере был установлен пакет BDE-5.

Программистам, использующим Delphi, этот скрипт понятен и не требует пояснений.

Теперь снова запустим Конфигуратор 1С:Предприятие, откроем обработку ВнешняяФорма-1_для_oleExecScript.ERT и внесем в текст процедуры `ВыполнитьСкрипт()` программного модуля следующие изменения (листинг 5.66).

**Листинг 5.66. Программный модуль обработки ВнешняяФорма-1_для_oleExecScript.ERT**

```
Процедура ВыполнитьСкрипт()

  СтрокаОшибки="";
  //-----------------------------
  // Выполнение скрипта
  Если oleExecScript.ExecuteBatch(ТекстСкрипта,0,СтрокаОшибки)>0 Тогда
     // Считываем значение переменной Count после
     // выполнения скрипта
     V = oleExecScript.GetVarValue("Count");
     Предупреждение("Значение переменной Count="+СокрЛП(V));
  Иначе
     Предупреждение("Ошибка! "+СтрокаОшибки);
  КонецЕсли;
  //-----------------------------

КонецПроцедуры
```

После успешного выполнения скрипта (`ExecuteBatch`) вызывается метод `GetVarValue`, куда в качестве параметра передается имя переменной `Count`.

Вновь запустим програму 1С:Предприятие, откроем обработку, введем скрипт и запустим его на выполнение (рис. 5.46).

Как видите, текст скрипта изменился по сравнению с предыдущим (см. рис. 5.45). А именно: объявлена еще одна переменная `Count: integer`, и результат запроса не выводится на экран (`ShowMessage`), а присваивается переменной `Count`.

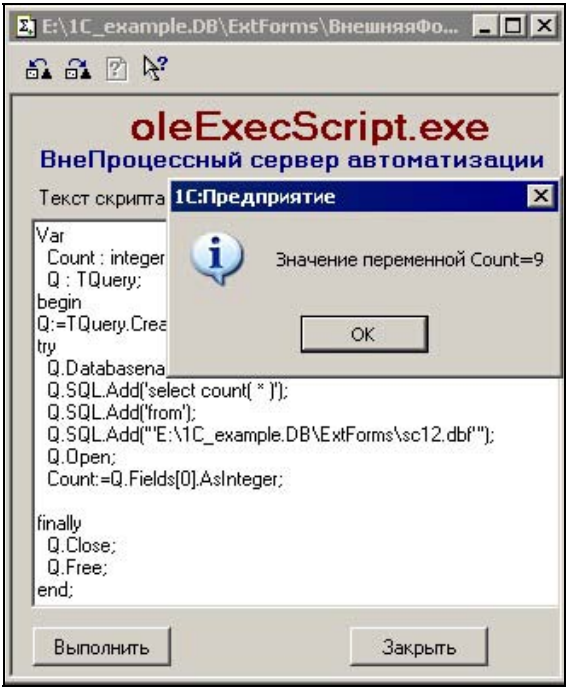



**Рис. 5.46.** Результат тестирования метода `GetVarValue` сервера oleExecScript.EXE

Сообщение "Значение переменной Count=9" (см. рис. 5.46) сгенерировано не скриптом, а программной строкой обработки (см. листинг 5.66):

```
Предупреждение("Значение переменной Count="+СокрЛП(V));
```

То есть на экран было выведено то, что вернул метод `GetVarValue` сервера автоматизации, *после* того как выполнился скрипт. С помощью методов сервера автоматизации oleExecScript.EXE можно не только выполнять скрипты различной сложности, но и считывать значения переменных скрипта после его выполнения, что очень существенно (особенно если учесть, что система 1С:Предприятие при использовании штатного метода `ExecuteBatch` не предоставляет такой возможности).

### *Примечание*

Программа oleExecScript.EXE была разработана "на скорую руку" (в течение полутора часов) только для демонстрационных целей и не претендует на большее.

## Часть II

# Доступ к базе данных 1С без программы 1С:Предприятие

В предыдущей части были рассмотрены вопросы обмена информаций (с программой 1:Предприятие) при помощи механизма OLE Automation. Как мы увидели, программа 1С:Предприятие обладает очень мощными и гибкими системными возможностями для доступа к информации в базе данных 1С (включая и метаданные) через механизм OLE Automation, позволяющими охватить весь спектр практических задач обмена информацией, стоящих перед разработчиками программного обеспечения. Но есть один существенный недостаток (относящийся, скорее, к механизму OLE Automation в целом) — довольно низкая скорость обмена информацией. В ряде случаев это может быть существенным ограничением для использования OLE Automation.

В настоящий момент для хранения базы данных в системе 1С:Предприятие предусмотрены два варианта: DBF-файлы и MS SQL Server (см. рис. 2.6). Это позволяет организовать доступ к информации в базах данных 1С (минуя программу 1С:Предприятие) посредством механизма стандартных SQL-запросов к таблицам баз данных, что в ряде случаев значительно эффективнее по быстродействию, чем доступ через механизм OLE Automation.

SQL (Structured Query Language) — универсальный язык структурированных запросов, ориентированный на операции с данными, представленными в виде логически взаимосвязанных совокупностей таблиц. Прототип языка (SEQUEL — произносится "сиквел") был создан компанией IBM в начале 1970-х. Позже язык SEQUEL был переименован в SQL. В 1986 году первый стандарт языка SQL был принят ANSI (American National Standards Institute) и официальным произношением стало "эс-кью-эл" ([,es kju:' el]), или "эс-ку-эль". За все время существования (в процессе развития) у языка было несколько стандартов. В настоящее время действует стандарт, принятый в 2003 году (SQL-3).

Одним из главных достоинств языка SQL является независимость от конкретной СУБД. А это значит, что с помощью SQL-запросов можно получать доступ как к локальным базам данных (DBF-файлы, MS Access и т. д.), так и к "серверным" базам данных (Oracle, Interbase, MS SQL Server и т. д.), используя общий подход в решении задач.

Забегая вперед, отмечу, что в рамках этой части я намеренно не буду касаться вопросов редактирования информации в базе данных 1С (без использования программы 1С:Предприятие). Это связано с тем, что модификация информации в рабочей (производственной) базе данных — дело весьма ответственное. Информации, представленной в этой части, достаточно для решения такой задачи. Но инструментарий, ориентированный на такую задачу, должен разработать программист под свою личную ответственность.

# Глава 6

# Файлы формата DBF

В 1980-е годы широкую популярность приобрела реляционная СУБД dBASE для персональных ЭВМ, созданная небольшой компанией Ashton Tate. Эта же компания предложила простой формат файлов для хранения реляционных таблиц, который получил название DBF. Как показало время, этот формат оказался очень "живучим" и ныне все еще не утратил своей актуальности как один из популярнейших форматов хранения информации при разработке локальных баз данных, а также как универсальный формат (по сути, принятый стандарт) обмена информацией между программными продуктами разных производителей.

## 6.1. Общая информация о DBF-формате

DBF-файл состоит из двух частей:

- полный заголовок (описатель структуры таблицы базы данных);
- блок информационных записей (собственно таблица базы данных).

Полный заголовок DBF-файла состоит из двух частей:

- общий заголовок таблицы (32 байта);
- блок описателей (дескрипторов) полей таблицы базы данных (размер каждого описателя — 32 байта).

Полный заголовок DBF-файла имеет размер $32 \times (N + 1)$ байт, где $N$ — количество полей в таблице базы данных. Полный заголовок DBF-файла отделяется от блока информационных записей *терминальным байтом* (ASCII-код 13). Файл завершается байтом с ASCII-кодом 26. Таким образом, блок информационных записей в файле начинается с байта номер $32 \times (N + 1) + 1$ при условии, что нумерация байтов начинается с нуля.

***Примечание***

Далеко не все разработчики программного обеспечения следуют тому правилу, что терминальный байт должен иметь ASCII-код 13. Очень часто там просто пробел (ASCII-код 32).

Все информационные записи в DBF-файле (строки таблицы базы данных) имеют строго определенный размер. Значения полей таблицы хранятся в ASCII-кодах (по сути, если исключить полный заголовок, это текстовые файлы).

Информационная запись DBF-файла состоит из двух частей:

- признак актуальности записи (помечена ли на удаление) — 1 байт;
- информационное содержание всех полей (кроме `MEMO` и `BLOB`) строки таблицы (без каких-либо разделителей).

Как видим, файлы DBF-формата имеют единую (в целом) структуру. Тем не менее, в зависимости от версии заголовки файлов могут отличаться по содержанию.

Рассмотрим структуру DBF-файлов на примере версии dBASE III+. В табл. 6.1 приведена структура общего заголовка таблицы базы данных.

***Таблица 6.1.*** *Структура общего заголовка таблицы для файлов формата dBASE III+*

| Номер байта | Число байт | Назначение |
|---|---|---|
| 0 | 1 | Признак наличия `MEMO`-поля:<br>• `03h` — `MEMO`-поля в таблице нет;<br>• `83h` — в таблице есть `MEMO`-поле (DBT-файл) |
| 1 | 3 | Дата последнего обновления таблицы в формате `YYMMDD`:<br>• 1-й байт — две младшие цифры года;<br>• 2-й байт — месяц;<br>• 3-й байт — день |
| 4 | 4 | Количество записей в таблице |
| 8 | 2 | Размер (в байтах) полного заголовка, включая терминальный байт |
| 10 | 2 | Размер информационной записи (в байтах) |
| 12 | 3 | Зарезервированная область |
| 15 | 13 | Зарезервированная область |
| 28 | 4 | Зарезервированная область |

**Примечание**

Далеко не все разработчики программного обеспечения следуют тому правилу, что байт номер 1 (общего заголовка таблицы) должен содержать две младшие цифры года даты последнего обновления файла.

В табл. 6.2 приведена структура описателя поля таблицы базы данных.

***Таблица 6.2.*** *Структура описателя поля таблицы базы данных в формате dBASE III+*

| Номер байта | Количество байт | Назначение |
|---|---|---|
| 0 | 10 | Имя поля таблицы базы данных (в ASCII-кодах). Если имя поля короче 11 символов, то "лишние" байты (справа) заполняются нулями |
| 11 | 1 | Тип поля:<br>`C` — строка (до 254 символов);<br>`N` — число (до 18 символов);<br>`D` — дата (8 символов);<br>`L` — логическое (1);<br>`M` — `MEMO`-поле |
| 12 | 4 | Адрес поля в памяти (ОЗУ). Вероятно, для внутренних целей СУБД, поддерживающих этот формат |
| 16 | 1 | Размер поля (в байтах) |
| 17 | 1 | Количество знаков после запятой (актуально только для полей типа `N`) |
| 18 | 14 | Зарезервированная область |

В табл. 6.3 приведена структура информационной записи DBF-файла или строки (возможно, только ее части) таблицы базы данных.

***Таблица 6.3.*** *Структура информационной записи таблицы базы данных в формате dBASE III+*

| Номер байта | Число байт | Назначение |
|---|---|---|
| 0 | 1 | Признак актуальности записи:<br>• если запись актуальна, то "пробел" (ASCII-код 32, или 20h);<br>• если запись помечена на удаление, то "звездочка" (ASCII-код 42, или 2Ah) |
| 1 | Сумма размеров полей таблицы | Значения полей таблицы (кроме `MEMO`-поля) |

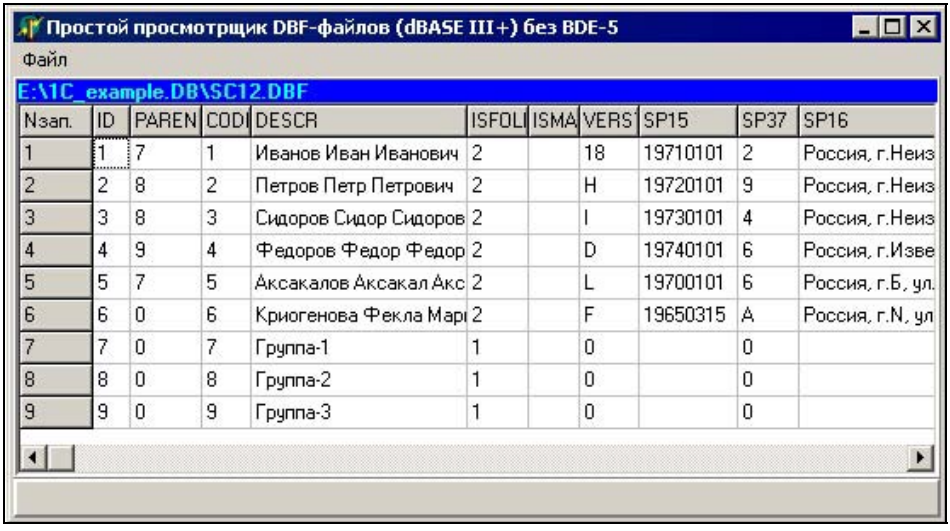
Очевидно, что термины "файл в формате DBF" и "таблица в формате DBF" в общем случае обозначают принципиально разные понятия. Например, строка таблицы базы данных может быть "распределена" больше, чем в одном файле. То есть значения всех полей (кроме `MEMO`) размещены в DBF-файле, а значения `MEMO`-полей — в DBT-файле. Ну и индексы, естественно.

## 6.2. Доступ к информации в DBF-файлах

В принципе, приложив определенные усилия, можно написать собственный инструментарий для доступа к информации, хранящейся в файлах формата DBF (без применения пакета BDE или других). Мы не будем здесь останавливаться на этом вопросе.

> ***Примечание***
>
> Папка Part_2\Chapter_06\Topic_2\sp_ViewDBF.pas\ на сопроводительном компакт-диске содержит исходные тексты примера простого просмотрщика небольших DBF-файлов.

На рис. 6.1 представлен результат открытия файла SC12.DBF (справочник `ФизЛица`) в программе sp_ViewDBF.exe.




| Nзап. | ID | PAREN | CODI | DESCR | ISFOLI | ISMA | VERS | SP15 | SP37 | SP16 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 7 | 1 | Иванов Иван Иванович | 2 | | 18 | 19710101 | 2 | Россия, г.Неиз |
| 2 | 2 | 8 | 2 | Петров Петр Петрович | 2 | | H | 19720101 | 9 | Россия, г.Неиз |
| 3 | 3 | 8 | 3 | Сидоров Сидор Сидоров | 2 | | I | 19730101 | 4 | Россия, г.Неиз |
| 4 | 4 | 9 | 4 | Федоров Федор Федор | 2 | | D | 19740101 | 6 | Россия, г.Изве |
| 5 | 5 | 7 | 5 | Аксакалов Аксакал Акс | 2 | | L | 19700101 | 6 | Россия, г.Б, ул. |
| 6 | 6 | 0 | 6 | Криогенова Фекла Мар | 2 | | F | 19650315 | A | Россия, г.N, ул |
| 7 | 7 | 0 | 7 | Группа-1 | 1 | | 0 | | 0 | |
| 8 | 8 | 0 | 8 | Группа-2 | 1 | | 0 | | 0 | |
| 9 | 9 | 0 | 9 | Группа-3 | 1 | | 0 | | 0 | |

**Рис. 6.1.** Простой просмотрщик DBF-файлов без использования BDE-5. Открыт файл SC12.DBF (справочник `ФизЛица`)

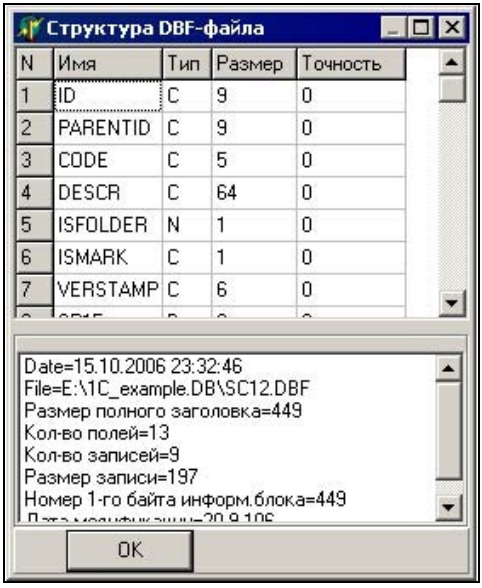



**Рис. 6.2.** Простой просмотрщик DBF-файлов без использования BDE-5. Структура открытого файла

На рис. 6.2 представлена структура открытого файла (информация прочитана из полного заголовка DBF-файла).

В особых случаях такой инструментарий может быть крайне полезен и даже незаменим. Тем не менее, в "обычной" жизни существенно проще (да и надежнее, пожалуй) использовать уже готовый инструментарий (например, пакет BDE-5). Хотя бы потому, что при решении главной задачи не придется отвлекаться на второстепенные вопросы. Поэтому в данной части мы (для организации доступа к информации в DBF-файлах) будем опираться на возможности пакета BDE-5.

Создадим новый Delphi-проект.

***Примечание***

Исходные тексты проекта sp_1Cv7_Viewer_Tables.DPR находятся на сопроводительном компакт-диске книги в папке \Part_2\sp_1Cv7_Viewer_Tables.PAS\001\.

На рис. 6.3 представлена форма проекта.

В листинге 6.1 представлен текст обработчика события "нажатие кнопки **Открыть таблицу**".

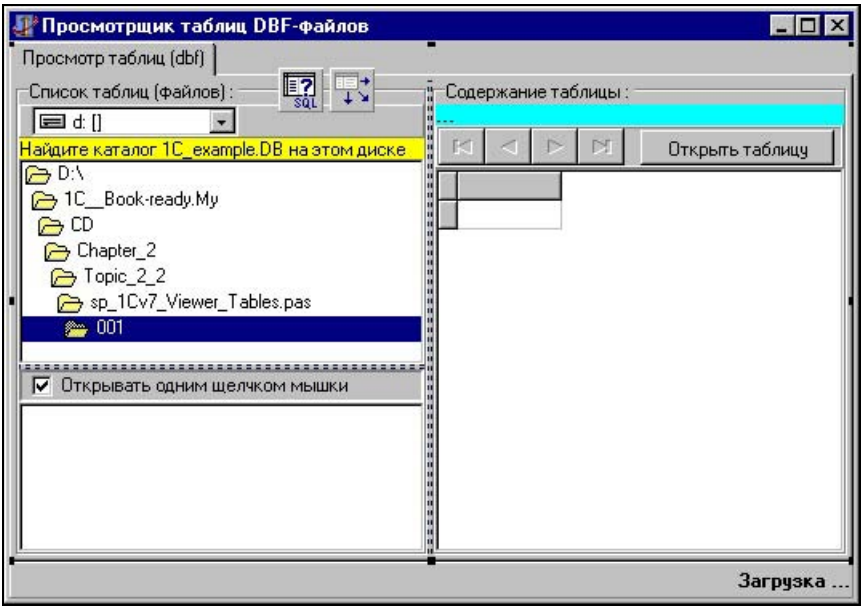



**Рис. 6.3.** Форма проекта sp_1Cv7_Viewer_Tables.DPR

### Листинг 6.1. Открытие DBF-файла для просмотра

```
procedure TForm1.SpeedButton1Click(Sender: TObject);
// Открытие DBF-файла для просмотра
begin
  Query1.Close;
  Query1.SQL.Clear;
  Label_TableName.Caption:='...';
  // Проверяем, выбран ли файл для просмотра
  if FileListBox1.ItemIndex>=0 then begin
     //-----------------------------
     // Формируем SQL-запрос для открытия DBF-фйла
     Query1.SQL.ADD('SELECT * ');
     Query1.SQL.ADD('FROM ');
     Query1.SQL.ADD(#39+FileListBox1.FileName+#39);
     //-----------------------------
     //Выполняем SQL-запрос
     Query1.Open;
     Label_TableName.Caption:=ExtractFileName(FileListBox1.FileName);
  end;
end;
```

Здесь все просто. Если в списке файлов (`FileListBox1`) выбран какой-либо DBF-файл, формируется и выполняется стандартный SQL-запрос (команда `SELECT`). После выполнения запроса на экране визуализируется информация из открытого набора данных.

Если сравнить тексты этого обработчика и программы sp_ViewDBF.EXE (для выполнения одного и того же действия), то видно, насколько проще организовать доступ к информации в формате DBF (и не только) при использовании пакета BDE-5 и соответствующих специальных компонентов из палитры компонентов Delphi.

Создадим еще одну диалоговую форму (рис. 6.4).

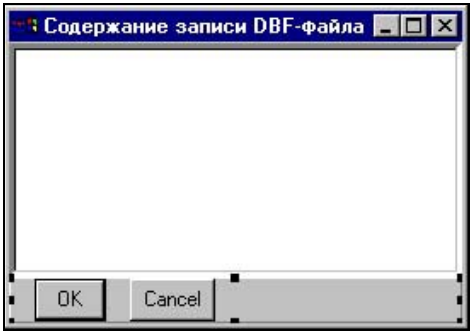



**Рис. 6.4.** Диалоговая форма `TDlg_View_Rec` для просмотра информационного содержания записи открытого набора данных

Ее назначение — просмотр информации из текущей записи открытого набора данных. Это полезно, если количество полей в открытом наборе данных может быть таким, что неудобно просматривать информационное содержание одной записи в табличной форме. В листинге 6.2 представлен текст метода `Execute` этой формы.

**Листинг 6.2. Просмотр записи открытого набора данных**

```
function TDlg_View_Rec.Execute(Q : TQuery) : boolean;
// Визуализация записи открытого набора данных
Var
  i : integer;
  Sx : string;
  Yes : byte;
begin
  Result:=FALSE;
```

```
  ListBox1.Items.Clear;

  if Q.Active then begin
     // Набор данных открыт для доступа
     //=====================================
     // Считываем информацию из текущей записи
     // открытого набора данных
     for i:=0 to (Q.Fields.Count-1) do
      begin
        // Имя поля
        Sx:=Q.Fields[i].FieldName+'=';
        Yes :=1;
        //---------------------
        // Проверяем тип поля.
        // Пропускаем поля следующих типов:
        //   MEMO
        if Q.Fields[i].DataType=ftMemo then Yes:=0;
        //   BLOB
        if Q.Fields[i].DataType=ftBlob then Yes:=0;
        //   Graphic
        if Q.Fields[i].DataType=ftGraphic then Yes:=0;
        // и т.д.
        //---------------------
        if Yes>0 then begin
           // Если "допустимое" поле - считываем значение
           Sx:=Sx+Q.Fields[i].AsString;
        end
        else begin
           // Если НЕ "допустимое" поле - пропускаем
           Sx:=Sx+'Это поле - не визуализируем';
        end;
        // Записываем сформированную строку
        // в свойство Items компонента ListBox1
        ListBox1.Items.Add(Sx);
     end;
     //=====================================
     // Показываем прочитанную информацию
     if ShowModal = mrOk then begin
        Result:=TRUE;
     end;
     ListBox1.Items.Clear;
  end;
end;
```

"Привяжем" вызов метода `Execute` формы `TDlg_View_Rec` к обработчику события "двойной щелчок мышкой на компоненте `DBGrid1`" (листинг 6.3).

**Листинг 6.3. Вызов метода `Execute` формы `TDlg_View_Rec`**

```
procedure TForm1.DBGrid1DblClick(Sender: TObject);
begin
  // Вызываем метод Execute формы T Dlg_View_Rec
  // для визуализации текущей записи открытого набора данных
  Dlg_View_Rec.Execute(Query1);
end;
```

Скомпилируем наш проект и запустим его на выполнение. Выберем файл SC12.DBF в каталоге базы данных 1C_example.DB, нажмем на кнопку **Открыть таблицу** и сделаем двойной щелчок левой кнопкой мышки на записи открытого набора данных, у которой значение поля `ID` равно 5 (компонент `DBGrid1`). Результат представлен на рис. 6.5.

Думаю, комментарий здесь не требуется.

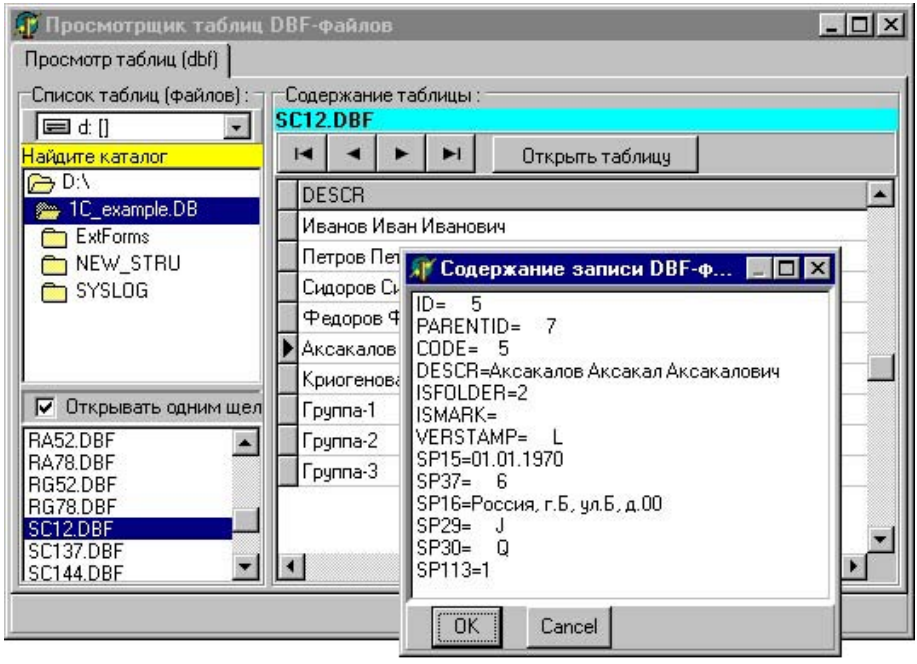



**Рис. 6.5.** Содержание записи открытого набора данных с `ID = 5`

# Глава 7

# Обработка словаря базы данных 1С (файл 1Cv7.DD)

В предыдущей главе мы создали простой просмотрщик DBF-файлов, позволяющий визуализировать содержание таблиц баз данных, хранящихся в формате DBF. Но применительно к базам данных 1С:Предприятие это малоэффективно.

Для того чтобы "расшифровать" информационное содержание таблиц, нужна дополнительная информация. В каждом каталоге базы данных 1С есть текстовый файл 1Cv7.DD (словарь данных), см. рис. 2.22. Посмотрим, каким образом можно использовать его для наших целей.

Откроем файл 1Cv7.DD в любом текстовом редакторе (например, в Notepad). Как видим, описание каждой таблицы представляет собой отдельный блок (см. рис. 2.29). Признаком начала блока может служить строка, начинающаяся с цепочки символов:

```
#==TABLE no
```

Признаком окончания блока может служить конец файла или строка с единственным символом:

```
#
```

Блоки, описывающие таблицы, разделены между собой строками:

```
#=========================================================
```

Каждый блок можно условно разделить на 3 части (рис. 7.1—7.3).

```
#==TABLE no 8       : Справочник ФизЛица
# Name     |Descr                          |Type[A/S/U]|DBTableName|ReUsable
T=SC12     |Справочник ФизЛица             |A          |SC12       |1
```

**Рис. 7.1.** Файл 1Cv7.DD. Блок описания таблицы. Описатель таблицы

```
#-----Fields-------
# Name       |Descr                |Type|Length|Precision
F=ID         |ID object            |C   |9     |0
F=PARENTID   |ID parent obj        |C   |9     |0
F=CODE       |object code          |C   |5     |0
F=DESCR      |object description   |C   |64    |0
F=ISFOLDER   |Flag - Is Line - Fol|N   |1     |0
F=ISMARK     |Flag Object is Marke|C   |1     |0
F=VERSTAMP   |Version stamp        |C   |6     |0
F=SP15       |(P)ДатаРождения      |D   |8     |0
F=SP37       |(P)ГородПроживания   |C   |9     |0
F=SP16       |(P)ДомашнийАдрес     |C   |64    |0
F=SP29       |(P)Пол               |C   |9     |0
F=SP30       |(P)СемейноеПоложение|C   |9     |0
F=SP113      |(P)СостоитВПрофсоюзе|N   |2     |0
```

**Рис. 7.2.** Файл 1Cv7.DD. Блок описания таблицы. Описатель полей таблицы

```
#----Indexes------
# Name      |Descr          |Unique|Indexed fields                    |DBName
I=IDD       |of ID          |0     |ID                                |IDD
I=PCODE     |of PARENT and  |0     |PARENTID,ISFOLDER,CODE(UPPER)     |PCODE
I=PDESCR    |of PARENT and  |0     |PARENTID,ISFOLDER,DESCR(UPPER)    |PDESCR
I=CODE      |of CODE        |0     |CODE(UPPER)                       |CODE
I=DESCR     |of DESCR       |0     |DESCR(UPPER)                      |DESCR
I=VI37      |VI37           |0     |SP37                              |VI37
I=VIP37     |VIP37          |0     |PARENTID,ISFOLDER,SP37            |VIP37
```

**Рис. 7.3.** Файл 1Cv7.DD. Блок описания таблицы. Описатель индексов таблицы

Интерпретируя описатель таблицы (рис. 7.1), можно извлечь имя таблицы (DBF-файла) из столбца `Name` или `DBTableName`, тип и вид (в контексте 1С:Предприятие) обязательно из первой строки описателя. В данном случае:

- имя таблицы (DBF-файла): `SC12`
- тип: `Справочник`
- вид (идентификатор): `ФизЛица`

***Примечание***

Столбцом `Descr` пользоваться "опасно" — по той причине, что информация в нем может быть усечена по ширине столбца.

Интерпретируя описатель полей таблицы (см. рис. 7.2), можно извлечь имя поля таблицы (DBF-файла) из столбца `Name` и его идентификатор (в контексте 1С:Предприятие) из столбца `Descr`.

Может оказаться полезной информация о формате этих полей:

- тип поля (столбец `Type`);
- длина (столбец `Length`);
- количество знаков после запятой (столбец `Precision`).

***Примечание***

Информация в столбце `Descr` может быть усечена по ширине столбца (рис. 7.4). В этом случае невозможно создать корректное соответствие между полем DBF-файла и его идентификатором в контексте 1С:Предприятие, пользуясь только файлом 1Cv7.DD.

```
F=VERSTAMP  |Version stamp        |C   |6     |0
F=SP139     |(P)Реквизит_Неопреде|C   |23    |0
F=TSP139    |                    |C   |3     |0
F=SP140     |(P)Реквизит_Справочн|C   |13    |0
F=SP141     |(P)Реквизит_Документ|C   |13    |0
F=SP142     |(P)Примечание       |C   |64    |0
```

**Рис. 7.4.** Файл 1Cv7.DD. Блок описания таблицы. Описатель полей таблицы. Усеченные идентификаторы

Тем не менее, попытаемся извлечь из файла 1Cv7.DD полезную информацию. Для начала попробуем выделить список таблиц (в данном случае DBF-файлов) базы данных 1С:Предприятие и создать соответствие между этими таблицами и их идентификаторами в контексте базы данных системы 1С:Предприятие.

Создадим новый Delphi-проект.

***Примечание***

Исходные тексты этого проекта (sp_1Cv7_DD_01.DPR) находятся на сопроводительном компакт-диске книги в папке \Part_2\Chapter_07\sp_1Cv7_DD_01.PAS\.

На рис. 7.5 представлена форма проекта.

Кнопка **Открыть 1Cv7.DD** предназначена для того, чтобы открыть файл 1Cv7.DD и загрузить его в свойство `Lines` компонента `RichEdit1`. Кнопка **Список таблиц** предназначена для того, чтобы создать список таблиц из файла 1Cv7.DD.

Здесь мы не будем углубляться в детали этого процесса. Листинг 7.1 представляет результат интерпретации файла 1Cv7.DD.

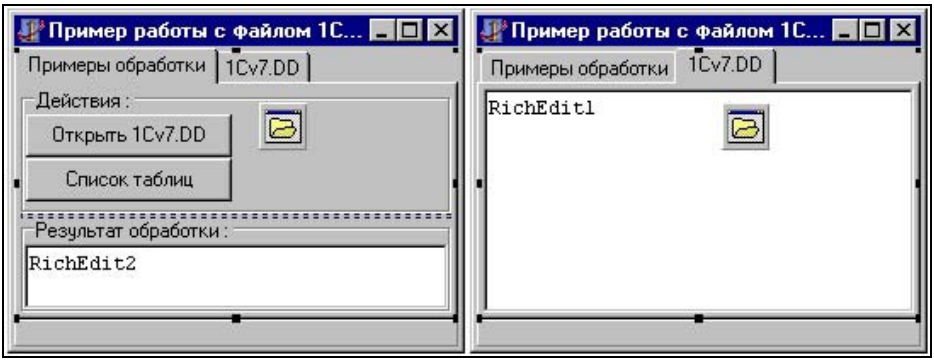



**Рис. 7.5.** Форма проекта sp_1Cv7_DD_01.DPR

**Листинг 7.1. Результат интерпретации файла 1Cv7.DD — список таблиц базы данных 1С:Предприятие**

```
Nстр_в_файле | Nтабл | ИмяТабл   | ДескрипторТабл
-----------------------------------------------
           7 |     0 |  1SUSERS  | Соединений
          16 |     1 | 1SSYSTEM  | Системная
          31 |     2 |  1SCONST  | Константы
          52 |     3 |  1SJOURN  | Журналы
          80 |     4 |  1SCRDOC  | Ссылки документов
          97 |     5 | 1SDNLOCK  | Номеров документов
         109 |     6 | 1SUIDCTL  | Уникальности
         121 |     7 |   1SBLOB  | Длинные строки
         135 |     8 |     SC12  | Справочник ФизЛица
         164 |     9 |     SC33  | Справочник СтраныМира
         181 |    10 |     SC35  | Справочник ГородаСтранМира
         201 |    11 |    SC137  | Справочник
                                    СправочникСНеопределеннымиРеквизитами
         223 |    12 |    SC144  | Справочник Новый1_Влад
         240 |    13 |    SC146  | Справочник Новый2_Подч
         262 |    14 |    SC155  | Справочник СправПериодРекв
         279 |    15 |     DH46  | Документ ИзменениеСостоянияФизЛиц
         292 |    16 |     DT46  | Документ (Мн.ч.) ИзменениеСостоянияФизЛиц
         306 |    17 |     DH88  | Документ ВыплатыФизЛицам
         319 |    18 |     DT88  | Документ (Мн.ч.) ВыплатыФизЛицам
         337 |    19 |     RG52  | Регистр УчетИзмененийСостоянияФизЛиц
```

```
        352 |     20 |      RA52 | Регистр (Дв.)
                                                УчетИзмененийСостоянияФизЛиц
        373 |     21 |      RG78 | Регистр УчетВыплатФизЛицам
        390 |     22 |      RA78 | Регистр (Дв.) УчетВыплатФизЛицам
-----------------------------------------------
```

Доработаем наш Delphi-проект.

***Примечание***

Исходные тексты этого примера (sp_1Cv7_DD_02.DPR) находятся на сопроводительном компакт-диске книги в папке \Part_2\Chapter_07\sp_1Cv7_DD_02.PAS\.

На рис. 7.6 представлена форма проекта.

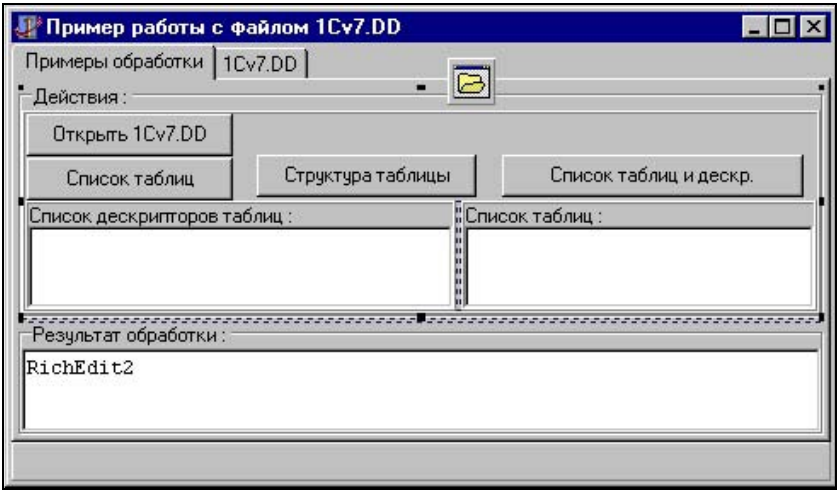



**Рис. 7.6.** Форма проекта sp_1Cv7_DD_02.DPR

Здесь по нажатию кнопки **Список таблиц** из открытого файла 1Cv7.DD выделяется список таблиц, который размещается следующим образом:

- идентификаторы таблиц (в контексте 1С:Предприятие) — в список **Список дескрипторов таблиц**;
- имена таблиц (DBF-файлов) — в список **Список таблиц**.

Оба списка синхронизированы в том смысле, что если выбирается строка в одном списке, то второй список автоматически позиционируется на соответствующей строке.

По нажатию кнопки **Структура таблицы** из открытого файла 1Cv7.DD выделяется информационный блок, описывающий выбранную таблицу, и результат помещается в свойство `Lines` компонента `RichEdit2` (рис. 7.7).

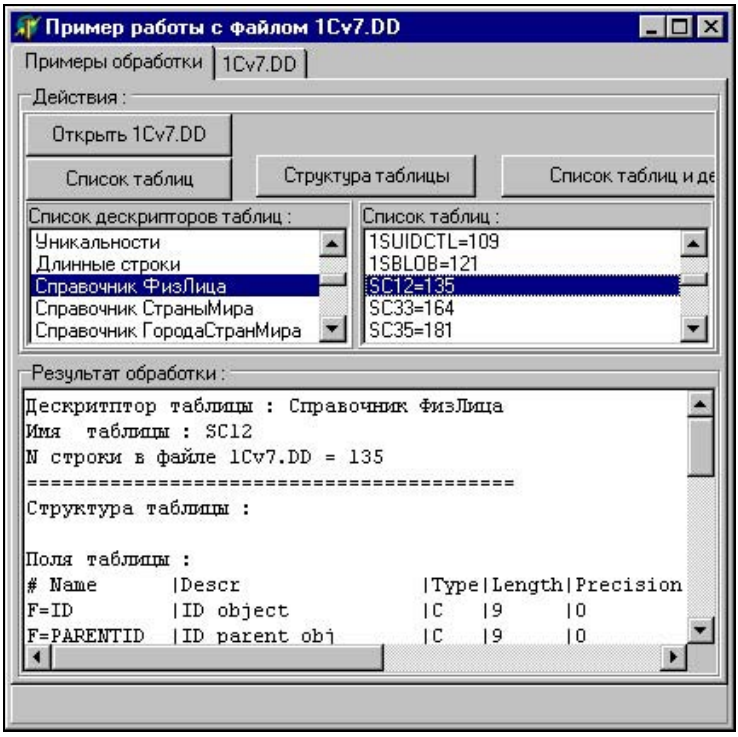



**Рис. 7.7.** Информационный блок, описывающий таблицу `SC12` (справочник `ФизЛица`)

По нажатию кнопки **Список таблиц и дескр.** из открытого файла 1Cv7.DD создается список таблиц (почти так же, как в предыдущем примере).

Мы не будем разбирать здесь исходные тексты этого проекта по причине тривиальности решения этой задачи.

# Глава 8

# Доступ к таблицам базы данных с учетом информации в файле 1Cv7.DD

Вернемся к нашему проекту sp_1Cv7_Viewer_Tables.DPR *(см. разд. 6.2)* и модифицируем его (с учетом обработки файла 1Cv7.DD).

***Примечание***

Исходные тексты этого примера находятся на сопроводительном компакт-диске книги в папке \Part_2\sp_1Cv7_Viewer_Tables.PAS\003\.

На рис. 8.1 представлена модифицированная форма проекта.

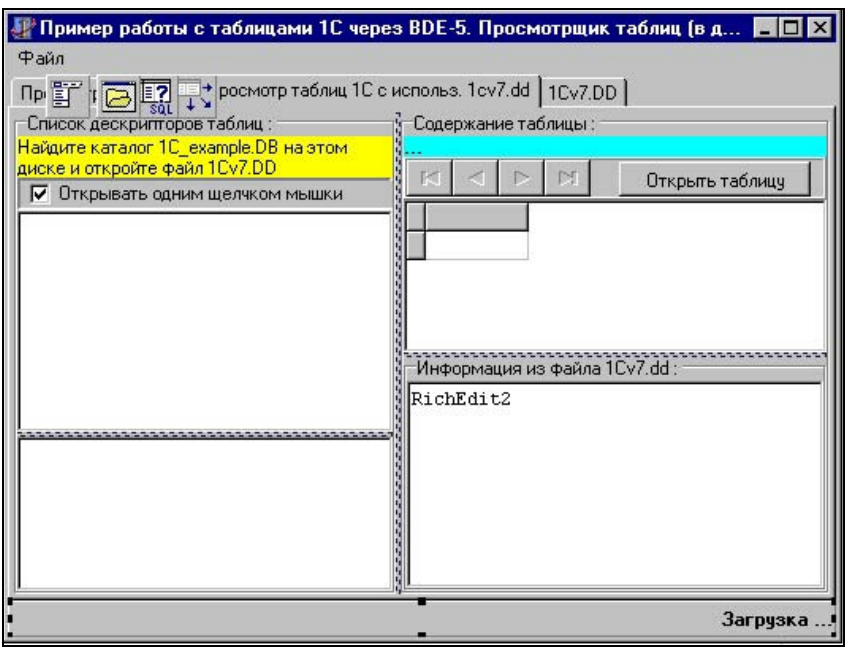



**Рис. 8.1.** Форма проекта sp_1Cv7_Viewer_Tables.DPR

Здесь, на вкладке **Просмотр таблиц 1С**, сохранены все возможности, имевшиеся в первоначальной версии проекта. Дополнительно:

- на вкладке **1Cv7.DD** — содержимое открытого файла 1Cv7.DD;
- на вкладке **Просмотр таблиц 1С с использ. 1cv7.dd** — просмотр DBF-файлов с учетом информации из файла 1Cv7.DD.

При выборе таблицы в списке, из файла 1Cv7.DD выделяется информационный блок (описатель выбранной таблицы), который (после некоторых несущественных преобразований) помещается в свойство `Lines` компонента `RichEdit2`. В дальнейшем при открытии таблицы эта информация используется для визуализации дескрипторов полей таблицы.

В листинге 8.1 представлен фрагмент обработчика события "нажатие кнопки **Открыть таблицу**".

**Листинг 8.1. Открытие DBF-файла для просмотра с учетом информации из файла 1Cv7.DD**

```
procedure TForm1.SpeedButton2Click(Sender: TObject);
Var
  ...
begin
 ...
if FileExists(fDBTableName_Full+'.dbf') then begin
    //----------------------------
    // Формируем SQL-запрос
    // и открываем набор данных
    Query1.SQL.ADD('SELECT * ');
    Query1.SQL.ADD('FROM ');
    Query1.SQL.ADD(#39+fDBTableName_Full+#39);
    Query1.Open;
    //----------------------------
    // Создаем переменную, где будем формировать
    // список полей DBF-файла и их дескрипторов
    // для визуализации
    ListFN:=TStringList.Create;
    TRY
      // Выделяем дескрипторы (наименования) полей
      // из фрагмента файла 1Cv7.DD для данной таблицы
      if GetFieldName_from_1Cv7DD(RichEdit2.Lines,
```

```
                                    ListFN)>0 then begin
         //-----------------------------
         // Показываем дескрипторы полей
         // (наименование в терминах 1С).
         // Для этого сканируем список полей открытого
         // набора данных Query1 и записываем в свойство
         // DisplayLabel соответствующий дескриптор
         for i:=0 to (Query1.Fields.Count-1) do
          begin
           // Выбираем из ListFN дескриптор, соответствующий
           // полю DBF-файла с именем FieldName
           Sx:=ListFN.VALUES[Query1.Fields[i].FieldName];
           // Записываем дескриптор в свойство DisplayLabel
           Query1.Fields[i].DisplayLabel:=Sx;
         end;
         //-----------------------------
      end;
    FINALLY
      if ListFN<>NIL then begin
         // Освобождаем ресурсы,
         // занятые переменной ListFN
         ListFN.Clear;
         ListFN.Free;
         ListFN:=NIL;
      end;
    END;
end
 else begin
    ShowMessage('Файл не найден : '+fDBTableName_Full);
 end;
 ...
end;
```

Если сравнить исходные тексты листингов 8.1 и 6.1, то видно: различаются они только тем, что после открытия набора данных (листинг 8.1) из файла 1Cv7.DD считываются дескрипторы полей таблицы (функция `GetFieldName_from_1Cv7DD`) и присваиваются свойству `DisplayLabel` соответствующего поля.

Назначение функции `GetFieldName_from_1Cv7DD`: обработка информационного блока (описателя заданной таблицы) с целью создания списка соответствий между именами полей таблицы и их дескрипторами (в контексте программы 1С:Предприятие). В листинге 8.2 приведен исходный текст этой функции.

**Листинг 8.2. Функция `GetFieldName_from_1Cv7DD`**

```
function GetFieldName_from_1Cv7DD(ListInfo,ListFN : TStrings) : integer;
// Выделение дескрипторов (наименований) полей из файла 1Cv7.DD:
// ListInfo - описатель таблицы из файла 1Cv7.DD (входной параметр),
// ListFN - дескрипторы полей (выходной параметр)
Var
  fn,             // Имя поля (DBF-файла)
  descr : string; // Дескриптор поля (наименование в терминах 1С)
  i : integer;
  Sx : string;
  InBlock : byte;
begin
 Result:=0;
 if ListFN<>NIL then begin
   ListFN.Clear;
   if ListInfo<>NIL then begin
     InBlock  := 0;
     i:=-1;
     while i<(ListInfo.Count-1) do
      begin
       i:=i+1;
       Sx:=Trim(ListInfo[i]);
       if length(Sx)>0 then begin
         if InBlock>0 then begin
           // Мы внутри блока описателя полей таблицы.
           // Теперь анализируем строку
           if Sx[1]='F' then begin
             // Эта строка - описатель поля.
             // Выделяем имя поля (DBF-файла)
             fn := Trim(sp_GetWordFromString('|' , Sx,1));
             fn := Trim(sp_GetWordFromString('=' , fn,2));
             // Выделяем дескриптор поля
```

```
            // (наименование в терминах 1С)
            descr := Trim(sp_GetWordFromString('|' , Sx,2));
            // Сохраняем в списке
            ListFN.VALUES[fn]:=descr;
          end;
        end;
        if POS('Поля таблицы :',Sx)>0 then begin
          // Внутри блока (описание полей таблицы)
          InBlock:=1;
        end;
        if POS('Индексы таблицы :',Sx)>0 then begin
          // Конец блока (описание полей таблицы)
          // Выходим из цикла
          InBlock:=0;
          i:=(ListInfo.Count+1);
        end;
      end;
    end;
  end;
  Result:=ListFN.Count;
 end;
end;
```

Здесь следует отметить, что на вход функции `GetFieldName_from_1Cv7DD` подается не вся информация из файла 1Cv7.DD, а уже выделенный блок (описатель таблицы) из свойства `Lines` компонента `RichEdit2` (см. рис. 8.1):

```
if GetFieldName_from_1Cv7DD(RichEdit2.Lines, ListFN)>0 then begin
```

Скомпилируем проект и запустим его на выполнение.

При выборе пункта меню **Открыть 1Cv7.DD** (рис. 8.2) нужно найти файл 1Cv7.DD в каталоге (например, 1C_example.DB) какой-либо базы данных 1С:Предприятие и открыть его.

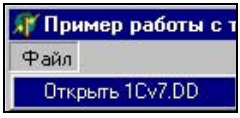



**Рис. 8.2.** Пункт меню для выбора файла 1Cv7.DD в каталоге базы данных 1С:Предприятие

Теперь откроем справочник `ФизЛица` и двойным щелчком мыши на строке таблицы с `ID` 5 откроем запись в отдельном окне. Результат представлен на рис. 8.3.

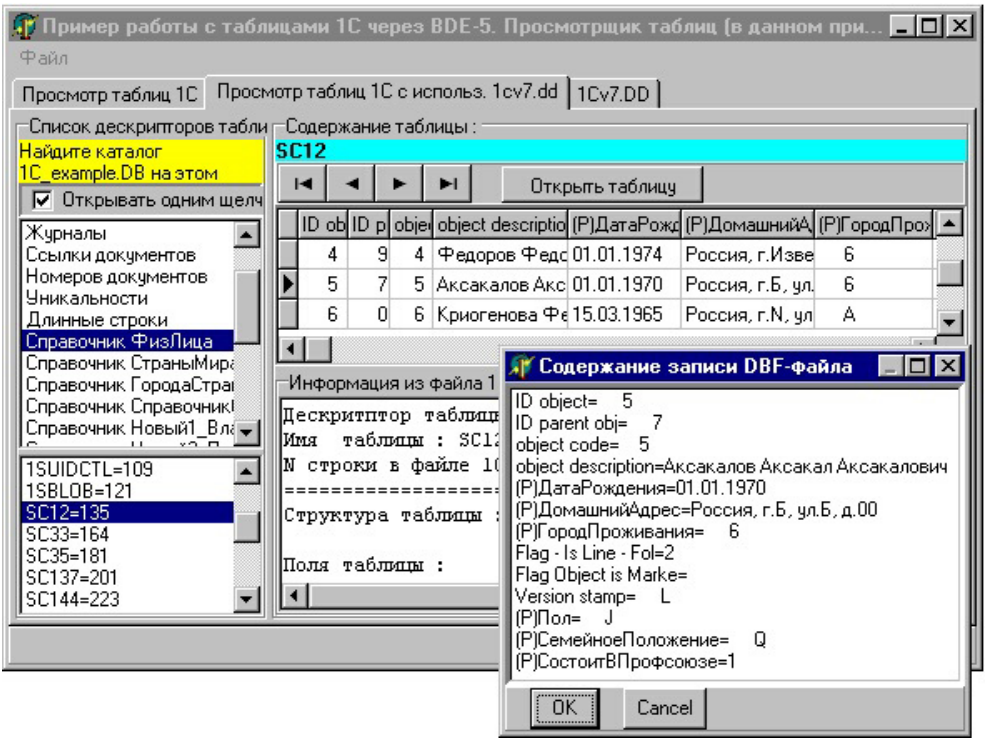



**Рис. 8.3.** Содержание записи с значением 5 поля `ID` открытого набора данных с учетом информации из файла 1Cv7.DD

### *Примечание*

Забегая вперед, отметим, что значения поля `ID` (так же, как и значения других полей таблиц базы данных, содержащих ссылки на объекты агрегатного типа, включая объекты метаданных), хранятся в 36-чной системе счисления. Исключение, пожалуй, составляют только ссылки на имена таблиц базы данных (например, поле `DNPREFIX` таблицы `1SJOURN`, см. далее.

Для сравнения на рис. 8.4 представлено информационное содержание элемента справочника (открытого в программе 1С:Предприятие), соответствующего данной записи таблицы.

В данном случае (см. рис. 8.3) мы открыли таблицу (DBF-файл), соответствующую справочнику `ФизЛица`. Если не считать полей типа `Строка неограниченной длины` и периодических реквизитов (которые хранятся в отдельных файлах),

вся информация из этого справочника содержится в одном DBF-файле. Но ситуация существенно усложняется, если речь идет о документах.

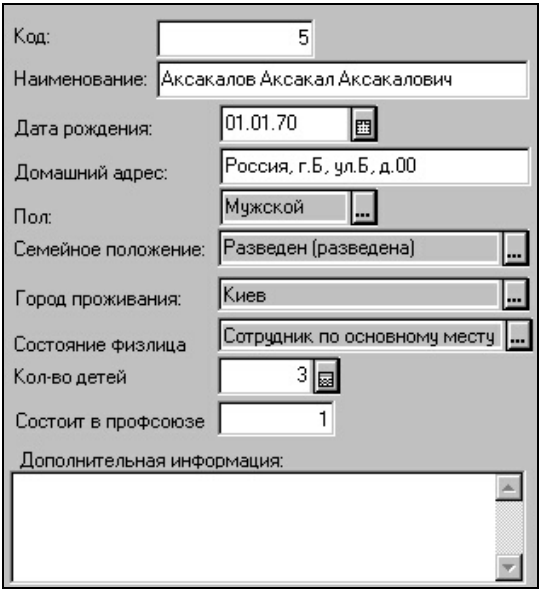



**Рис. 8.4.** Содержание элемента (с кодом 5) справочника `ФизЛица` (программа 1С:Предприятие)

Для примера рассмотрим один из документов вида `ВыплатыФизЛицам` (номер 1, дата 14.07.2006, время 12:00:00).

Запустим программу 1С:Предприятие и откроем журнал документов (рис. 8.5).

| | Номер | Дата | Время | Документ |
|---|---|---|---|---|
| | 1 | 28.06.06 | 23:38:42 | Изменение состояния физ.лиц |
| | 3 | 29.06.06 | 12:26:28 | ВыплатыФизЛицам |
| | 2 | 12.07.06 | 23:26:40 | Изменение состояния физ.лиц |
| | 3 | 12.07.06 | 23:29:02 | Изменение состояния физ.лиц |
| | 1 | 14.07.06 | 12:00:00 | ВыплатыФизЛицам |
| | 2 | 15.07.06 | 11:53:53 | ВыплатыФизЛицам |

**Рис. 8.5.** Журнал документов, открытый в программе 1С:Предприятие

Если мы откроем файл 1SJOURN.DBF (соответствует журналу документов) в нашем "просмотрщике" (рис. 8.6), то увидим, что все далеко не так красиво, как на рис. 8.5.

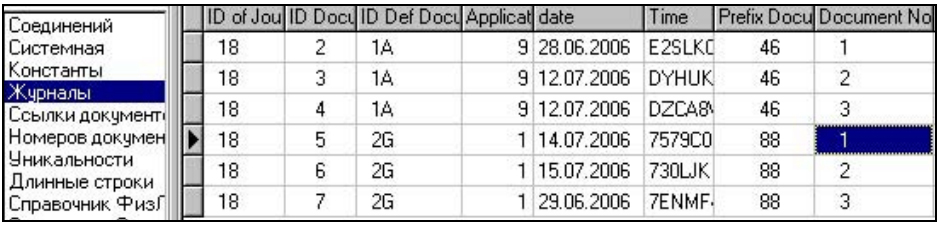



**Рис. 8.6.** Журнал документов, открытый в программе sp_1Cv7_Viewer_Tables.EXE

В табл. 8.1 приведено соответствие имен полей файла 1SJOURN.DBF (важных с точки зрения идентификации документов) и их дескрипторов из файла 1Cv7.DD.

***Таблица 8.1.*** *Соответствие имен полей файла 1SJOURN.DBF и их дескрипторов из файла 1Cv7.DD*

| Дескриптор поля | Имя поля | Значение | Примечание |
|---|---|---|---|
| `ID Document` | `IDDOC` | **5** | Идентификатор документа |
| `date` | `DATE` | 14.07.2006 | Дата документа |
| `Prefix Document No` | `DNPREFIX` | **88** | Номер в имени DBF-файла (в нашем случае), соответствует свойству `Вид` документов |
| `Document No` | `DOCNO` | 1 | Номер документа |

Теперь откроем документ в программе 1С:Предприятие (рис. 8.7).

Если мы попробуем открыть этот документ в нашем "просмотрщике", то окажется, что это, мягко говоря, сложно сделать, потому что этому виду документов соответствуют две таблицы (т. е. два DBF-файла):

- D**H**88.DBF — реквизиты шапки документа и общие реквизиты документов;
- D**T**88.DBF — реквизиты табличной части документа.

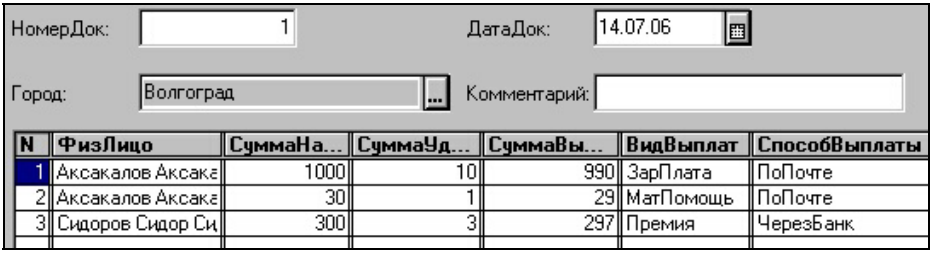



**Рис. 8.7.** Документ открыт в программе 1С:Предприятие

Здесь номер 88 в имени DBF-файла соответствует значению поля `DNPREFIX` файла 1SJOURN.DBF (в нашем случае). То есть открывать этот документ придется "по частям" — отдельно каждый файл (что не очень хорошо для пользователя).

Тем не менее, откроем файл DH88.DBF (рис. 8.8).

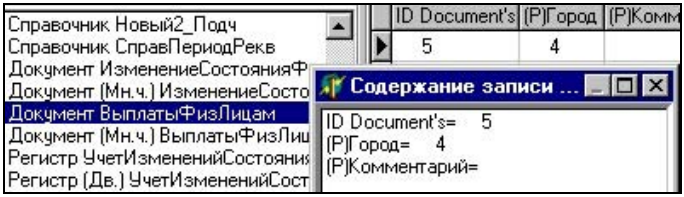



**Рис. 8.8.** "Шапка" документа вида `ВыплатыФизЛицам` (открыта в программе sp_1Cv7_Viewer_Tables.EXE)

Как видим, в этом файле не хранится ни номер документа, ни дата документа.

Теперь откроем файл DT88.DBF (рис. 8.9).

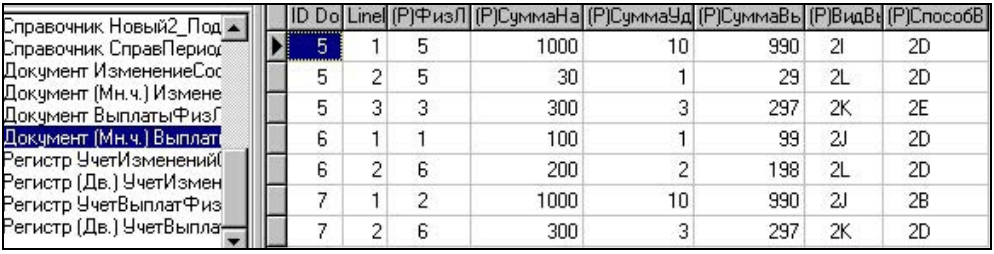



**Рис. 8.9.** Табличная часть документов вида `ВыплатыФизЛицам` (открыта в программе sp_1Cv7_Viewer_Tables.EXE)

Как видим, здесь собраны все строки табличной части всех документов данного вида. То есть нужно фильтровать эту таблицу по идентификатору документа (поле `IDDOC`), чего наш просмотрщик делать еще не умеет.

Таким образом, можно сделать вывод: для того чтобы открыть документ как целостный объект (по аналогии с программой 1С:Предприятие), нужно прочитать информацию, как минимум, из трех DBF-файлов:

- 1SJOURN.DBF — журнал документов (чтобы получить номер и дату документа);
- DH*.DBF — "шапка" документов (чтобы получить значения реквизитов "шапки" документа и общих реквизитов документов);
- DT*.DBF — табличная часть документов.

При этом нужно выполнить следующие действия:

1. Открыть файл 1SJOURN.DBF, выбрать интерактивно (или найти по идентификатору) нужный документ и прочитать значения полей `DNPREFIX`, `DOCNO`, `DATE` и `IDDOC`.
2. Открыть файл DH*.DBF (вместо звездочки должны быть первые 10 символов значения поля `DNPREFIX` файла 1SJOURN.DBF), найти по полю `IDDOC` соответствующий документ и прочитать значения всех полей записи.
3. Открыть файл DT*.DBF (вместо звездочки должно быть значение поля `DNPREFIX` файла 1SJOURN.DBF), отфильтровать по полю `IDDOC` все записи (строки) соответствующего документа и прочитать значения всех полей из каждой записи (строки).
4. "Расшифровать" значения реквизитов агрегатного типа (в рамках данной главы мы это не рассматриваем).

***Внимание!***

Поле `DNPREFIX` может содержать, например, и такие значения:

`"303702002"`

`" 25943"`

***Примечание***

В любом случае под системный идентификатор вида документа выделены первые 10 символов (включая пробелы).

На сопроводительном компакт-диске книги в папке \Part_2\sp_1Cv7_Viewer_Tables.PAS\004\ приведены исходные тексты примера, иллюстрирующего этот алгоритм открытия документа (в упрощенном виде, естественно). Здесь мы не будем вдаваться в подробности реализации (кто желает — посмотрит сам) по той причине, что с точки зрения простого считывания информации из таблиц базы данных с использованием BDE-5 там ничего сложного нет. А с точки зрения "расшифровки" значений реквизитов агрегатного типа (что было бы интересно), там тоже ничего нет.

На рис. 8.10 показан результат визуализации документа `ВыплатыФизЛицам` с номером 1 от 14.07.2006.

Здесь следует отметить, что информации, содержащейся в файле 1Cv7.DD, явно не хватает, чтобы корректно выявить тип реквизитов и "расшифровать" значения реквизитов агрегатного типа (а также получить значения периодических реквизитов). Например, если в справочник `ФизЛица` внести еще один

реквизит (идентификатор — `строка9`, тип — `Строка`, длина — `9`), то в файле 1Cv7.DD это отразится (для файла SC12.DBF) следующим образом:

```
F=SP37        |(P)ГородПроживания   |C    |9      |0
...
F=SP166       |(P)строка9           |C    |9      |0
```

То есть невозможно корректно различить реквизиты `ГородПроживания` (агрегатный тип) и `строка9` (базовый тип) по типу.

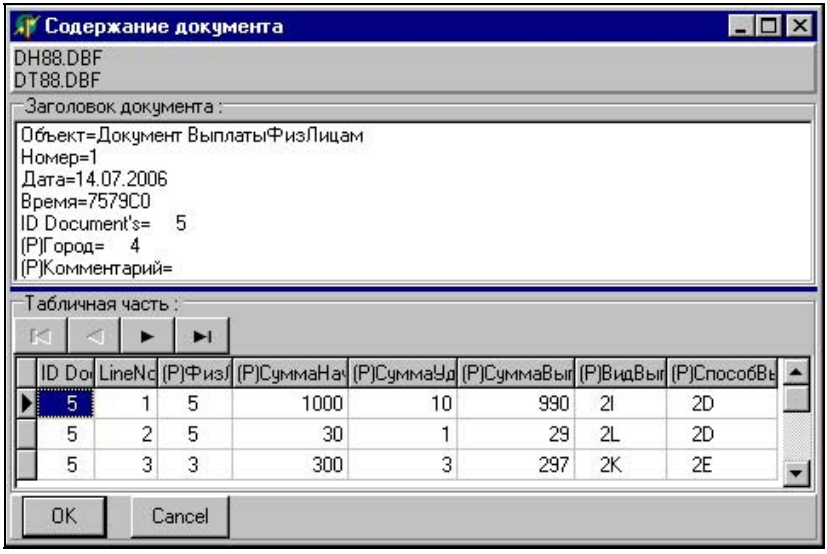



**Рис. 8.10.** Документ `ВыплатыФизЛицам` (с номером 1) от 14.07.2006 (открыт в программе sp_1Cv7_Viewer_Tables.EXE)

# Глава 9

# Дополнительные данные об информационном содержании таблиц, полученные с помощью механизма OLE Automation

В *разд. 4.4* мы получили соответствия между идентификаторами значений перечислений и их системными (внутренними) идентификаторами (см. рис. 4.183). Попробуем использовать эту информацию в нашем просмотрщике.

Вспомним, что в программе sp_LoadStructure_1C.exe есть специальная кнопка , предназначенная для сохранения информационной структуры 1С:Предприятие в специальном текстовом INI-файле (описателе структуры). Здесь мы не будем вдаваться в подробности реализации этого алгоритма (желающие могут посмотреть исходные тексты программы). Если открыть результирующий файл 1C_example_DB.TXT и найти раздел `Перечисления_СоответствияЗначений_2`, то можно увидеть, что содержание этого раздела (листинг 9.1) может помочь нам при визуализации полей DBF-файлов, значения которых имеют тип `Перечисление`.

**Листинг 9.1. Содержание раздела `Перечисления_СоответствияЗначений_2`**

```
[Перечисления_СоответствияЗначений_2]
~Направленность=SysID->ID
J=Мужской
K=Женский
M=НикогдаНеСостоялВБраке
N=СостоитВЗарегистрированномБраке
O=СостоитВНеЗарегистрированномБраке
P=ГражданскийБрак
Q=Разведен
```

```
R=Разошелся
S=Вдовец
13=СотрудникОсновной
14=СотрудникНеосновной
15=ВременноНеРаботает
20=Высокий
21=Обычный
22=Низкий
2B=ЧерезКассу
2E=ЧерезБанк
2D=ПоПочте
2I=ЗарПлата
2J=Аванс
2K=Премия
2L=МатПомощь
```

Используя эту информацию, можно при визуализации значений полей открытого набора данных (в нашем случае — таблиц из DBF-файлов) "расшифровать" значения полей, соответствующих агрегатному типу `Перечисление` (в контексте 1С:Предприятие). Проблема в том, что невозможно идентифицировать такие поля, основываясь только на информации из файла 1Cv7.DD.

Тем не менее (чисто для иллюстративных целей), попробуем реализовать эту возможность в нашем проекте. Прежде чем визуализировать значение любого поля, которое является реквизитом (справочника, документа или регистра), проверим, присутствует ли это значение среди параметров списка соответствий (листинг 9.1). И если присутствует — предполагаем (в общем случае, необоснованно), что это значение есть `Перечисление`.

> ***Примечания***
>
> Подобный подход категорически недопустим при решении практических задач!

Проверяем, является ли поле таблицы реквизитом, по двум признакам:

- если имя поля таблицы начинается с символов `SP`, это реквизит;
- если имя поля таблицы равно `VALUE`, это реквизит (значение периодического реквизита).

То есть так:

```
if (POS('SP', AnsiUpperCase(Q.Fields[i].FieldName))=1) or
  ('VALUE' = AnsiUpperCase(Q.Fields[i].FieldName)) then begin
  // Предполагаем, что это реквизит
```

```
  // (или значение периодического реквизита)
  ...
end;
```

В листинге 9.2 приведен исходный текст функции, возвращающей идентификатор значения перечисления по его системному (внутреннему) значению.

**Листинг 9.2. Функция чтения идентификатора значения перечисления по его системному идентификатору**

```
function TDlg_View_Record.DecodeEnumValue(sVal : string) : string;
// Поиск идентификатора значения перечисления
// по его системному (внутреннему) идентификатору:
// sVal - системный (внутренний) идентификатор
Const
  TopicName = 'Перечисления_СоответствияЗначений_2';
Var
  // fn - имя INI-файла, где находится раздел
  // [Перечисления_СоответствияЗначений_2]
  fn : string;
begin
  Result:='';
  sVal:=Trim(sVal);
  if length(sVal)>0 then begin
     // Имя INI-файла с OLE-структурой
     fn:=Trim(Form1.FileName_oleStr);
     if length(fn)>0 then begin
        if FileExists(fn) then begin
           // Считываем идентификатор перечисления:
           Result:=Trim(sp_GetFromINI_StringParam(fn,TopicName,sVal,''));
        end;
     end;
  end;
end;
```

Напомню, что назначение функции `sp_GetFromINI_StringParam` (разработанной нами ранее) — считывание информации из INI-файлов.

***Примечание***

На сопроводительном компакт-диске книги в папке \Part_2\sp_1Cv7_Viewer_Tables.PAS\005\ приведены исходные тексты новой версии проекта. Подробно рассматривать эти тексты здесь нет смысла.

На рис. 9.1 приведен пример визуализации записи из файла SC12.DBF (справочник `ФизЛица`).

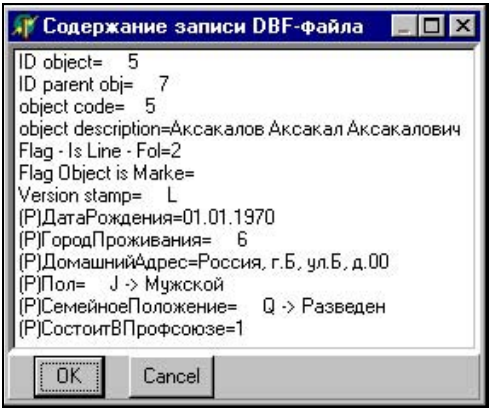



**Рис. 9.1.** Визуализация записи из файла SC12.DBF

На рис. 9.2 приведен пример визуализации записи из файла DT88.DBF (табличная часть документов вида `ВыплатыФизЛицам`).

На рис. 9.3 приведен пример визуализации записи из файла RA78.DBF (движение по регистрам вида `УчетВыплатФизЛицам`).

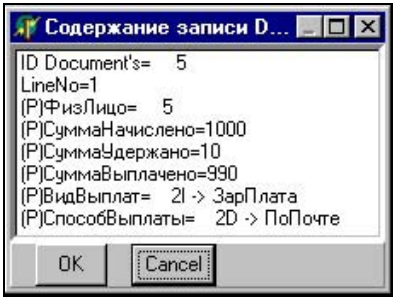



**Рис. 9.2.** Визуализация записи из файла DT88.DBF

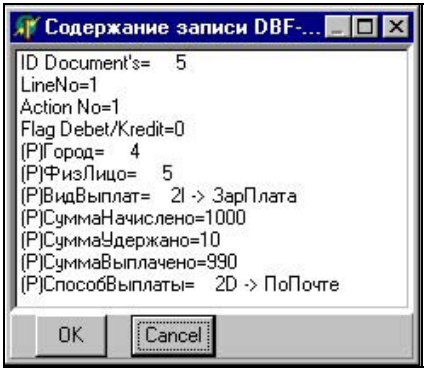



**Рис. 9.3.** Визуализация записи из файла RA78.DBF

# Глава 10

# Дополнительные данные об информационном содержании таблиц, полученные из файла 1Cv7.MD

В предыдущих главах мы столкнулись с тем, что для доступа к базам данных 1С (вне контекста программы 1С:Предприятие) не хватает информации из файла 1Cv7.DD, позволяющей корректно вычислять значения реквизитов агрегатного типа. Этот факт делает невозможным создание универсального программного обеспечения, ориентированного на доступ к информационному содержанию баз данных 1С (вне контекста программы 1С:Предприятие).

В *разд. 4.4* мы рассматривали возможность считывания информационной структуры базы данных 1С из конфигурационного файла 1Cv7.MD.

> ***Примечание***
>
> Напомню, что в папках Part_1\Chapter_04\Topic_4_4\Item_1_4_4\ и \Part_2\sp_1Cv7_Viewer_Tables.PAS\006\ на сопроводительном компакт-диске книги находится файл 1Cv7_md_Main_MetaData_Stream.TXT с информацией о структуре, прочитанной из потока `Main MetaData Stream` файла 1Cv7.MD.

На рис. 10.1 представлен фрагмент информационного содержания файла 1Cv7_md_Main_MetaData_Stream.TXT.

После экспериментов (с использованием Конфигуратора 1С и программы Read_from_1Cv7_MD.EXE), которые может провести любой желающий, и анализа информационного содержания потока `Main MetaData Stream` было выявлено следующее: в строках, соответствующих справочникам, количество "членов" = 18 (каждый "член" заключен в двойные кавычки, разделитель — запятая). Их назначение приведено в табл. 10.1.

```
{"SbCnts",
{"12","ФизЛица","Пример","Список физических лиц","0","5","2","2","2","64","1","1","
{"Params",
{"15","ДатаРождения","","Дата рождения","D","0","0","0","0","0","0","1","0","0","1"
{"42","СостояниеФизлица","","Состояние физлица","E","0","0","38","0","0","1","1","0
{"43","КолвоДетей","","Количество детей","N","10","0","0","0","0","1","1","0","0","
{"37","ГородПроживания","","Город проживания","B","0","0","35","0","0","0","1","0",
{"16","ДомашнийАдрес","","Домашний адрес","S","64","0","0","0","0","0","1","0","0",
{"29","Пол","","Пол","E","0","0","17","0","0","0","1","0","0","1","0","0"},
{"30","СемейноеПоложение","","Семейное положение","E","0","0","21","0","0","0","1",
{"113","СостоитВПрофсоюзе","","Состоит в профсоюзе","N","1","0","0","0","0","0","1"
{"31","ДополнительнаяИнформация","","Дополнительная информация","S","0","0","0","0"
```

**Рис. 10.1.** Фрагмент информационного содержания файла 1Cv7_md_Main_MetaData_Stream.TXT

***Таблица 10.1.*** *"Расшифровка" строк потока* `Main MetaData Stream`*, соответствующих справочникам*

| Номер "члена" в строке | Назначение |
|---|---|
| 1 | Системный идентификатор. Остается постоянным даже при изменении идентификатора (вида) справочника в Конфигураторе 1С |
| 2 | Идентификатор (вид) справочника (как он задан в Конфигураторе) |
| 3 | Комментарий |
| 4 | Синоним |
| 5 | Системный идентификатор владельца |
| 6 | Длина кода элемента справочника |
| 7 | Серии кодов (2 — в пределах подчинения, 1 — во всем справочнике) |
| 8 | Тип кода (2 — числовой, 1 — текстовый) |
| 9 | Автоматическая нумерация (2 — да, 1 — нет) |
| 10 | Длина наименования элемента справочника |
| 11 | Основное представление (1 — в виде наименования, 0 — в виде кода) |
| 12 | Редактировать (0 — в списке, 1 — в диалоге, 2 — обоими способами) |
| 13 | Количество уровней |
| 14 | Системный идентификатор формы списка |
| 15 | Системный идентификатор формы списка |
| 16 | Одна форма для элемента и группы (1 — да, 0 — нет) |
| 17 | Контроль уникальности (1 — да, 0 — нет) |
| 18 | Размещать группы сверху (1 — да, 0 — нет) |

В строках, соответствующих реквизитам справочников, количество "членов" — 17. Назначение некоторых из них приведено в табл. 10.2.

***Таблица 10.2.*** *"Расшифровка" строк потока* `Main MetaData Stream`*, соответствующих реквизитам справочников*

| Номер "члена" в строке | Назначение |
|---|---|
| 1 | Системный идентификатор. Остается постоянным даже при изменении идентификатора (или типа) реквизита в Конфигураторе 1С |
| 2 | Идентификатор реквизита (как он задан в Конфигураторе) |
| 3 | Комментарий |
| 4 | Синоним |
| 5 | Тип:<br>`S` — строка<br>`N` — число<br>`D` — дата<br>`U` — неопределенный<br>`B` — справочник<br>`O` — документ<br>`E` — перечисление<br>`T` — счет<br>`K` — субконто<br>`P` — план счетов<br>`C` — календарь<br>`A` — вид расчета |
| 6 | Длина (если тип = `S`, то длина = 0 означает, что это строка неограниченной длины) |
| 7 | Число знаков после запятой |
| 8 | Если тип агрегатный (и вид определенный), то системный идентификатор (ссылка на соответствующий справочник, документ, перечисление и т. д.) |
| 9 | Неотрицательный (1 — да, 0 — нет) |
| 10 | Разделять на триады (1 — да, 0 — нет) |
| 11 | Периодический (1 — да, 0 — нет) |
| 12 | Использовать для элемента (1 — да, 0 — нет) |
| 13 | Использовать для группы (1 — да, 0 — нет) |
| 14 | Сортировка (1 — да, 0 — нет) |
| 15 | Если периодический:<br>ручное изменение (1 — да, 0 — нет) |
| 16 | Если периодический: изменяется документами (1 — да, 0 —нет) |

В строках, соответствующих документам, количество "членов" — 15. Назначение некоторых из них приведено в табл. 10.3.

***Таблица 10.3.*** *"Расшифровка" строк потока* `Main MetaData Stream`*, соответствующих документам*

| Номер "члена" в строке | Назначение |
|---|---|
| 1 | Системный идентификатор. Остается постоянным даже при изменении идентификатора (вида) документа в Конфигураторе 1С |
| 2 | Идентификатор (вид) документа (как он задан в Конфигураторе) |
| 3 | Комментарий |
| 4 | Синоним |
| 5 | Длина номера |
| 6 | Периодичность:<br>0 — по всем документам данного вида<br>1 — в пределах года<br>2 — в пределах квартала<br>3 — в пределах месяца<br>4 — в пределах дня |
| 7 | Тип номера (2 — числовой, 1 — текстовый) |
| 8 | Автоматическая нумерация (2 — да, 1 — нет) |
| 9 | Системный идентификатор журнала |
| 11 | Контроль уникальности (1 — да, 0 — нет) |
| 13 | Оперативный учет (1 — да, 0 — нет) |
| 14 | Расчет (1 — да, 0 — нет) |

В строках, соответствующих реквизитам "шапки" документов, количество "членов" — 10. В строках, соответствующих реквизитам табличной части документов, количество "членов" — 11. Их назначение приведено в табл. 10.4.

***Таблица 10.4.*** *"Расшифровка" строк потока* `Main MetaData Stream`*, соответствующих реквизитам документов*

| Номер "члена" в строке | Назначение |
|---|---|
| 1 | Системный идентификатор. Остается постоянным даже при изменении идентификатора (или типа) реквизита в Конфигураторе 1С |
| 2 | Идентификатор реквизита (как он задан в Конфигураторе) |
| 3 | Комментарий |
| 4 | Синоним |
| 5 | Тип:<br>`S` — строка<br>`N` — число<br>`D` — дата<br>`U` — неопределенный<br>`B` — справочник<br>`O` — документ<br>`E` — перечисление<br>`T` — счет<br>`K` — субконто<br>`P` — план счетов<br>`C` — календарь<br>`A` — вид расчета |
| 6 | Длина |
| 7 | Число знаков после запятой |
| 8 | Если тип агрегатный (и вид определенный), то системный идентификатор (ссылка на соответствующий справочник, документ, перечисление и т. д.) |
| 9 | Неотрицательный (1 — да, 0 — нет) |
| 10 | Разделять на триады (1 — да, 0 — нет) |
| 11 | Если это реквизит табличной части, то итог по столбцу (1 — да, 0 — нет) |

В строках, соответствующих регистрам, количество "членов" — 8. Назначение некоторых из них приведено в табл. 10.5.

***Таблица 10.5.*** *"Расшифровка" строк потока* `Main MetaData Stream`*, соответствующих регистрам*

| Номер "члена" в строке | Назначение |
|---|---|
| 1 | Системный идентификатор. Остается постоянным даже при изменении идентификатора (вида) регистра в Конфигураторе 1С |
| 2 | Идентификатор (вид) регистра (как он задан в Конфигураторе) |
| 3 | Комментарий |
| 4 | Синоним |
| 5 | Тип (0 — остатки, 1 — обороты) |
| 6 | Периодичность (`M` — месяц, `Q` — квартал, `Y` — год) |
| 7 | Быстрая обработка движения (1 — да, 0 — нет) |

В строках, соответствующих измерениям регистров, количество "членов" — 12. В строках, соответствующих реквизитам регистров, количество "членов" — 11. В строках, соответствующих ресурсам регистров, количество "членов" — 10. Их назначение приведено в табл. 10.6.

***Таблица 10.6.*** *"Расшифровка" строк потока* `Main MetaData Stream`*, соответствующих реквизитам (измерениям, ресурсам) регистров*

| Номер "члена" в строке | Назначение |
|---|---|
| 1 | Системный идентификатор. Остается постоянным даже при изменении идентификатора (или типа) реквизита, измерения или ресурса в Конфигураторе 1С |
| 2 | Идентификатор реквизита (как он задан в Конфигураторе) |
| 3 | Комментарий |
| 4 | Синоним |
| 5 | Тип:<br>`S` — строка<br>`N` — число<br>`D` — дата<br>`U` — неопределенный<br>`B` — справочник<br>`O` — документ<br>`E` — перечисление<br>`T` — счет<br>`K` — субконто<br>`P` — план счетов<br>`C` — календарь<br>`A` — вид расчета |

***Таблица 10.6*** *(окончание)*

| Номер "члена" в строке | Назначение |
|---|---|
| 6 | Длина |
| 7 | Число знаков после запятой |
| 8 | Если тип агрегатный (и вид определенный), то системный идентификатор (ссылка на соответствующий справочник, документ, перечисление и т. д.) |
| 9 | Неотрицательный (1 — да, 0 — нет) |
| 10 | Разделять на триады (1 — да, 0 — нет) |
| 11 | Если измерения или ресурсы, то отбор движений (1 — да, 0 — нет) |
| 12 | Если измерения, то отбор итогов (1 — да, 0 — нет) |

В строках, соответствующих перечислениям, количество "членов" — 4. Их назначение приведено в табл. 10.7.

***Таблица 10.7.*** *"Расшифровка" строк потока* `Main MetaData Stream`*, соответствующих перечислениям*

| Номер "члена" в строке | Назначение |
|---|---|
| 1 | Системный идентификатор. Остается постоянным даже при изменении идентификатора (вида) перечисления в Конфигураторе 1С |
| 2 | Идентификатор (вид) перечисления (как он задан в Конфигураторе) |
| 3 | Комментарий |
| 4 | Синоним |

В строках, соответствующих значениям перечислений, количество "членов" — 5. Их назначение приведено в табл. 10.8.

***Таблица 10.8.*** *"Расшифровка" строк потока* `Main MetaData Stream`*, соответствующих значениям перечислений*

| Номер "члена" в строке | Назначение |
|---|---|
| 1 | Системный идентификатор. Остается постоянным даже при изменении идентификатора значения перечисления в Конфигураторе 1С |
| 2 | Идентификатор значения перечисления (как он задан в Конфигураторе) |
| 3 | Комментарий |
| 4 | Представление |
| 5 | Представление |

Теперь можно сделать некоторые выводы.

- Просмотрев файл 1Cv7_md_Main_MetaData_Stream.TXT, можно сделать вывод, что все объекты конфигурации 1С:Предприятие имеют сквозную нумерацию системных идентификаторов, которая никак не зависит от типа объекта или его "принадлежности" к другим объектам конфигураци.
- Просмотрев файлы 1Cv7.DD и 1Cv7_md_Main_MetaData_Stream.TXT, можно сделать вывод, что все поля таблиц (DBF-файлов), которые являются реквизитами (непериодическими), ресурсами или измерениями (начинаются с символов `SP`), содержат в имени число, которое однозначно соотносится с системным идентификатором объекта конфигурации 1С (первый "член" в строках файла 1Cv7_md_Main_MetaData_Stream.TXT).
- Если открыть файл 1SCONST.DBF, где хранятся значения периодических реквизитов справочников, и провести определенный анализ, то видно, что значения поля `ID` — это те же системные идентификаторы объектов из файла 1Cv7_md_Main_MetaData_Stream.TXT, но представленные в 36-ричной системе счисления.
- Если открыть файл 1SBLOB.DBF, где хранятся значения реквизитов типа `Строка неограниченной длины`, то точно такие же выводы (как и с периодическими реквизитами) можно сделать относительно значений поля `FIELDID`.
- Все значения полей, имеющих тип `Перечисление`, так же однозначно соотносятся с системными идентификаторами объектов конфигурации 1С и тоже представлены в 36-ричной системе счисления.

Примеры:

1. В табл. 10.9 представлены реквизиты справочника `ФизЛица`, созданные в разное время. Очевидно, что системные идентификаторы были присвоены им в порядке очередности, наряду с другими объектами конфигурации.

***Таблица 10.9.*** *Реквизиты справочника* `ФизЛица`

| **Реквизит** | **Системный идентификатор** | **Представление в 36-ричной системе счисления** | **Имя поля в файле SC12.DBF** |
|---|---|---|---|
| ДатаРождения | 15 | F | SP15 |
| СостояниеФизлица | 42 | 16 | — |
| КолвоДетей | 43 | 17 | — |
| ГородПроживания | 37 | 11 | SP37 |
| ДомашнийАдрес | 16 | G | SP16 |
| Пол | 29 | T | SP29 |
| СемейноеПоложение | 30 | U | SP30 |
| СостоитВПрофсоюзе | 113 | 35 | SP113 |
| ДополнительнаяИнформация | 31 | V | — |

2. В табл. 10.9 показано соответствие полей файла SC12.DBF (который соответствует справочнику `ФизЛица`) и системных идентификаторов реквизитов справочника `ФизЛица`.
3. В табл. 10.10 показано соответствие значений поля `ID` файла 1SCONST.DBF (значения периодических реквизитов справочников) и системных идентификаторов реквизитов `СостояниеФизлица` и `КолвоДетей` (представленных в 36-ричной системе счисления).

***Таблица 10.10.*** *Периодические реквизиты и реквизиты типа* `Строка неограниченной длины` *справочника* `ФизЛица`

| **Реквизит** | **Системный идентификатор** | **Представление в 36-ричной системе счисления** | **Значение поля `ID` в файле 1SCONST.DBF** | **Значение поля `FIELDID` в файле 1SBLOB.DBF** |
|---|---|---|---|---|
| СостояниеФизлица | 42 | 16 | 16 | — |
| КолвоДетей | 43 | 17 | 17 | — |
| ДополнительнаяИнформация | 31 | V | — | V |

4. В табл. 10.10 показано соответствие значений поля `FIELDID` файла 1SBLOB.DBF (значения реквизитов типа `Строка неограниченной длины`) и системного идентификатора реквизита `ДополнительнаяИнформация` (представленного в 36-ричной системе счисления).
5. В табл. 10.11 показано соответствие системных идентификаторов значений перечислений (представленных в 36-ричной системе счисления) и значений полей DBF-файлов.

***Таблица 10.11.*** *Соответствие системных идентификаторов значений перечислений (представленных в 36-ричной системе счисления) и значений полей DBF-файлов*

| Значение перечисления | Системный идентификатор | Представление в 36-ричной системе счисления | Файл, поле, значение |
|---|---|---|---|
| `Пол.Мужской` | 19 | J | SC12, SP29, J |
| `Пол.Женский` | 20 | K | SC12, SP29, K |
| `СемейноеПоложение.Разведен` | 26 | Q | SC12, SP30, Q |
| `СостояниеФизлица.СотрудникОсновной` | 39 | 13 | DT46, SP51, 13 |
| `ВидВыплатФизЛицам.МатПомощь` | 93 | 2L | RA78, SP103, 2L |
| `СостояниеФизлица.ВременноНеРаботает` | 41 | 15 | 1SCONST, VALUE, 15 (для ID = 16) |

Попробуем использовать полученную выше информацию для наших целей.

Рассмотрим процесс визуализации DBF-файлов (не путать с объектами базы данных 1С). Алгоритм визуализации значений полей открытых DBF-файлов может быть следующим (кроме файлов 1SBLOB.DBF и 1SCONST.DBF):

1. Считывается значение очередного поля, имя которого начинается с символов `SP`.

2. Из имени поля выделяется его системный идентификатор (как объекта 1С).
3. Определяются `Тип` и `Вид` объекта (информация из 1Cv7.MD).
4. Если тип `Базовый` — значение визуализируется.
5. Если тип `Агрегатный`, то вычисляется значение (в зависимости от типа) и визуализируется.

Алгоритм визуализации значений поля `VALUE` файла 1SCONST.DBF:

1. Считывается значение поля `ID` (системный идентификатор в 36-ричной системе счисления).
2. Считывается значение поля `VALUE` (значение реквизита).
3. Значение поля `ID` конвертируется в десятичную систему счисления.
4. Определяется `Тип` и `Вид` объекта (информация из 1Cv7.MD).
5. Если тип `Базовый` — визуализируется значение поля `VALUE`.
6. Если тип `Агрегатный`, то вычисляется значение (в зависимости от типа) и визуализируется.

С визуализацией файла 1SBLOB.DBF (реквизиты типа `Строка неограниченной длины`) дело обстоит иначе. Сначала должен быть задан соответствующий реквизит справочника, значение которого визуализируется, и выбран элемент справочника, для которого визуализируется значение реквизита. Затем устанавливается соответствующий фильтр. Фильтр формируется с учетом того, что системный идентификатор реквизита должен быть преобразован из десятичной системы счисления в 36-ричную. Набор данных сортируется по полю `BLOCKNO`, информация (значение поля `BLOCK`) выбирается поочередно, начиная с первой записи набора данных, и объединяется в одной переменной (соответствующего типа).

Удобства ради (и упрощения для) сведем информацию о структуре базы данных из файлов 1Cv7.DD и 1Cv7_md_Main_MetaData_Stream.TXT в единый файл, по структуре подобный INI-файлу. Из файла 1Cv7.DD будем использовать список таблиц (DBF-файлов) и список полей для каждой таблицы (листинг 10.1).

**Листинг 10.1. Формат хранения информации (INI-файл). Список DBF-файлов**

```
[ТАБЛИЦЫ]
1SCONST=Константы
1SBLOB=Длинные строки
SC12=Справочник ФизЛица
```

```
...
DH88=Документ ВыплатыФизЛицам
DT88=Документ (Мн.ч.) ВыплатыФизЛицам
...
RG78=Регистр УчетВыплатФизЛицам
RA78=Регистр (Дв.) УчетВыплатФизЛицам
...
[SC12]
ID=1
PARENTID=1
CODE=1
DESCR=1
ISFOLDER=1
ISMARK=1
VERSTAMP=1
SP15=1
SP37=1
SP16=1
SP29=1
SP30=1
SP113=1
```

Список таблиц нужен для того, чтобы уже не обращаться к файлу 1Cv7.DD при визуализации списка таблиц в приложении.

Список полей по каждой таблице нужен для того, чтобы можно было достаточно быстро контролировать наличие некоторых полей, которые могут участвовать в вычислении значений объектов агрегатного типа, но иной раз отсутствуют. Например, поле `DESCR` (в файлах SC*.DBF) соответствует свойству `Наименование` элементов справочников. Иной раз (хоть и редко) оно может отсутствовать. Можно, конечно, в INI-файле проверять значение параметра `ДлинаНаименования` для объекта, соответствующего этому имени файла (таблице). Если значение равно нулю, то свойство `Наименование` у элементов справочника отсутствует. Но список полей таблиц может быть полезен (попутно) и для других целей. Например, для визуализации имен столбцов. В этом случае формат строки может быть таким:

```
SP15=Дата рождения
```

Из файла 1Cv7_md_Main_MetaData_Stream.TXT будем использовать все объекты (справочники, документы, регистры, реквизиты, ресурсы, измерения и т. д.), которые представляют интерес с точки зрения извлечения и визуализа-

ции информации из базы данных. Каждому объекту будет соответствовать отдельный раздел INI-файла (листинг 10.2):

**Листинг 10.2. Формат хранения информации (INI-файл) об объектах конфигурации 1С**

```
[17]
Идентификатор=Пол
ЧтоЭто=Перечисление
ИдентификаторСистемный=17
ИдентификаторСистемный36=H
Синоним=Пол

[19]
Идентификатор=Мужской
ЧтоЭто=ЗначениеПеречисления
Родитель=17
ИдентификаторСистемный=19
ИдентификаторСистемный36=J
Синоним=Мужской

[12]
Идентификатор=ФизЛица
ЧтоЭто=Справочник(ТаблицаБД)
ИдентификаторСистемный=12
ИдентификаторСистемный36=C
Синоним=Список физических лиц
Тип=Справочник
Вид=ФизЛица
КоличествоУровней=2
ДлинаКода=5
СерииКодов=2
ТипКода=2
ДлинаНаименования=64
ОсновноеПредставление=1

[42]
Идентификатор=СостояниеФизлица
ЧтоЭто=РеквизитСправочника(ПолеТаблицыБД)
```

```
Родитель=12
ИдентификаторСистемный=42
ИдентификаторСистемный36=16
Синоним=Состояние физлица
Периодический=1
Тип=Перечисление
Вид=38
Длина=0
Точность=0
Неотрицательный=0
РазделятьНаТриалы=0
Сортировка=0
РучноеИзменение=1
ИзменяетсяДокументами=1
```

Теперь, наконец, приступим к программированию. За основу возьмем наш предыдущий проект (удалив из него все лишнее).

> ***Примечание***
>
> Исходные тексты этого примера находятся на сопроводительном компакт-диске книги в папке \Part_2\sp_1Cv7_Viewer_Tables.PAS\006\.

Прежде чем программировать доступ к базе данных 1С, нужно сформировать INI-файл, где будет храниться объединенная структура базы данных (листинг 10.2).

Извлечение списка таблиц (DBF-файлов) и их полей из файла 1Cv7.DD мы подробно рассмотрели в *главе 8* и здесь на этом останавливаться не будем. А вот выделение нужной информации из файла 1Cv7_md_Main_MetaData_Stream.TXT (рис. 10.1) полезно будет затронуть подробнее.

Строки такого формата проще всего, пожалуй, обрабатывать с помощью свойства `CommaText` Delphi-объекта `TStrings`. В листинге 10.3 приведен исходный текст функции, реализующей этот способ.

**Листинг 10.3. Обработка строк в SDF-формате**

```
function sp_Convert_SDF_to_List(Sx : string; List : TStrings) : integer;
// Разделение строки Sx (в SDF-формате) на составляющие:
// List — результат (в каждой строке списка — отдельный "член")
begin
  Result:=0;
```

```
  if List<>NIL then begin
     List.Clear;
     Sx:=sp_AllTrimUnChar('{}, ',Sx);
     if length(Sx)>0 then begin
        List.CommaText:=Sx;
     end;
     Result:=List.Count;
  end;
end;
```

Назначение функции `sp_AllTrimUnChar` — удаление не-символов из строки слева и справа. Текст ее прост и здесь не приводится.

В дальнейшем удобно считывать значения "членов" (согласно их назначению, см. табл. 10.1—10.8) из списка `TStrings` с помощью простой функции (листинг 10.4).

**Листинг 10.4. Считывание информации из списка `TStrings`**

```
function GetValueFromList(NumParam: integer; List : TSTrings) : string;
// Считывание иформации из списка (по номеру параметра):
// NumParam — номер "члена" (начиная с 1),
// List — результат функции sp_Convert_SDF_to_List
Var
  Index : integer;
begin
  Result:='';
  if List<>NIL then begin
     Index:=NumParam-1;
     if (Index>=0) and (Index<List.Count) then begin
        Result:=Trim(List[Index]);
     end;
  end;
end;
```

Пример использования:

```
if sp_Convert_SDF_to_List(Sx, ListBuf)>0 then begin
    // Считываем значение свойства "Длина кода
    // элемента справочника" справочника
```

```
S1:=GetValueFromList(6, ListBuf);
V1:=sp_StrToInt(S1,0);
ListINI.ADD('ДлинаКода='+IntToStr(V1));
```

Здесь после считывания строки, соответствующей справочнику, вызывается функция `sp_Convert_SDF_to_List`. Затем из списка `ListBuf` считывается (согласно табл. 10.1) значение свойства `Длина кода элемента справочника` (функция `GetValueFromList`).

Для удобства обработки (и большей наглядности) файлы такого формата (см. рис. 10.1) можно преобразовать в "дерево" с помощью символов табуляции (рис. 10.2).

```
{"SbCnts",
        {"12","ФизЛица","Пример","Список физических лиц","0","5","2","2","2","64"
                {"Params",
                        {"15","ДатаРождения","","Дата рождения","D","0","0","0","
                        {"42","СостояниеФизлица","","Состояние физлица","E","0","
                        {"43","КолвоДетей","","Количество детей","N","10","0","0"
                        {"37","ГородПроживания","","Город проживания","B","0","0"
                        {"16","ДомашнийАдрес","","Домашний адрес","S","64","0","0
                        {"29","Пол","","Пол","E","0","0","17","0","0","0","1","0"
                        {"30","СемейноеПоложение","","Семейное положение","E","0"
                        {"113","СостоитВПрофсоюзе","","Состоит в профсоюзе","N","
                        {"31","ДополнительнаяИнформация","","Дополнительная инфор
                {"Form",
                        {"13","ФормаСписка","","}}},
        {"33","СтраныМира","","Страны мира","0","5","2","2","2","64","1","0","1",
                {"Params"},
                {"Form",
                        {"34","ФормаСписка","","}}},
```

**Рис. 10.2.** Фрагмент информационного содержания файла 1Cv7_md_Main_MetaData_Stream.TXT в виде "дерева"

В листинге 10.5 приведен исходный текст функции, реализующей это преобразование.

**Листинг 10.5. Формирование содержания файла 1Cv7_md_Main_MetaData_Stream.TXT в виде "дерева"**

```
function sp_Convert_Info_from_1Cv7MD_To_TreeText(
                                        List_1Cv7MD : TStrings) : integer;
// Преобразование информации из потока Main MetaData Stream
// в виде "дерева" (не нарушая структуры строк):
// List_1Cv7MD - входной и выходной параметр
// (содержание файла 1Cv7_md_Main_MetaData_Stream.TXT)
```

```
Var
  CountTab, // Номер уровня иерархии
  i : integer;
  S1,
  Sx : string;
  List : TStrings;
begin
 Result:=0;
 if List_1Cv7MD<>NIL then begin
   if List_1Cv7MD.Count>0 then begin
     // Создаем буферную переменную
     List:=TStringList.Create;
     TRY
      CountTab:=-1;
      for i:=0 to (List_1Cv7MD.Count-1) do
       begin
       Sx:=Trim(List_1Cv7MD[i]);
       S1:=Sx;
       // Определяем, сколько в строке слева символов "{",
       // и увеличиваем уровень иерархии для текущей строки
       CountTab:=CountTab + sp_GetCountCharInString(S1,ORD('{'));
       // Смещаем вправо (согласно уровню иерархии)
       Sx:=sp_Append_Tab_to_StringLeft(Sx,CountTab);
       List.Add(Sx);
       // Определяем, сколько в строке всего символов "}",
       // и уменьшаем уровень иерархии для следующей строки
       CountTab:=CountTab - sp_GetCountCharInString(S1,ORD('}'));
       Application.ProcessMessages;
      end;
      // Возвращаем результат преобразования
      List_1Cv7MD.Assign(List);
      Result:=List.Count;
     FINALLY
      // Освобождаем ресурсы
      if List<>NIL then begin
         List.Clear;
         List.Free;
         List:=NIL;
```

```
      end;
     END;
   end;
 end;
end;
```

Идея проста. Если в строке встречается символ `{`, то нужно сдвинуть всю текущую строку вправо на один знак табуляции (следующий уровень иерархии). Если в строке встречается символ `}`, то нужно сдвинуть всю следующую строку влево на один знак табуляции (предыдущий уровень иерархии).

В листинге 10.6 приведены тексты функций `sp_GetCountCharInString` и `sp_Append_Tab_to_StringLeft`.

**Листинг 10.6. Функции `sp_GetCountCharInString` и `sp_Append_Tab_to_StringLeft`**

```
function sp_GetCountCharInString(Sx : string; CodeChar : byte) : integer;
// Подсчет символов с ASCII-кодом CodeChar в строке
Var
  i : integer;
begin
  Result:=0;
  if length(Sx)>0 then begin
     i:=0;
     while i<length(Sx) do
      begin
        i:=i+1;
        if ORD(Sx[i])=CodeChar then begin
           Result:=Result+1;
        end;
     end;
  end;
end;

function sp_Append_Tab_to_StringLeft(Sx : string;
                                     CountTab : integer) : string;
// Добавление в начало строки заданного числа символов табуляции
Var
```

```
  i : integer;
begin
  Result:=Sx;
  if CountTab>0 then begin
     for i:=1 to CountTab do
      begin
        Result:=#9+Result;
     end;
  end;
end;
```

В результате преобразования информационного потока `Main MetaData Stream` (файла 1Cv7.MD) на первом уровне иерархии (один знак табуляции в начале строки) оказываются такие узлы "дерева", как:

`{"Consts",` — константы

`{"SbCnts",` — справочники

`{"Registers",` — регистры

`{"Documents",` — документы

`{"EnumList",` — перечисления

и др. По сути, это агрегатные типы данных (в контексте 1С).

Дочерними узлами (относительно перечисленных выше узлов) на следующем уровне иерархии (два знака табуляции в начале строки) оказываются объекты, представляющие собой виды указанных выше агрегатных типов данных. То есть это списки справочников, регистров, документов, перечислений). Каждый вид (объектов) тоже является узлом "дерева". Он включает в себя разделы информации (три знака табуляции в начале строки). И, наконец, каждый раздел (как узел "дерева") включает в себя "листья дерева" (четыре знака табуляции в начале строки).

"Листьями дерева" являются, например:

- реквизиты справочников и документов;
- измерения, ресурсы и реквизиты регистров;
- значения перечислений.

Схематично это отражено на рис. 10.3—10.6.

Теперь для обработки "дерева" (после преобразования информационного потока `Main MetaData Stream` из файла 1Cv7.MD) легко можно разработать соответствующий алгоритм выделения нужной информации.

**Рис. 10.3.** Ветвь "дерева" от узла `{"SbCnts",` (справочники)

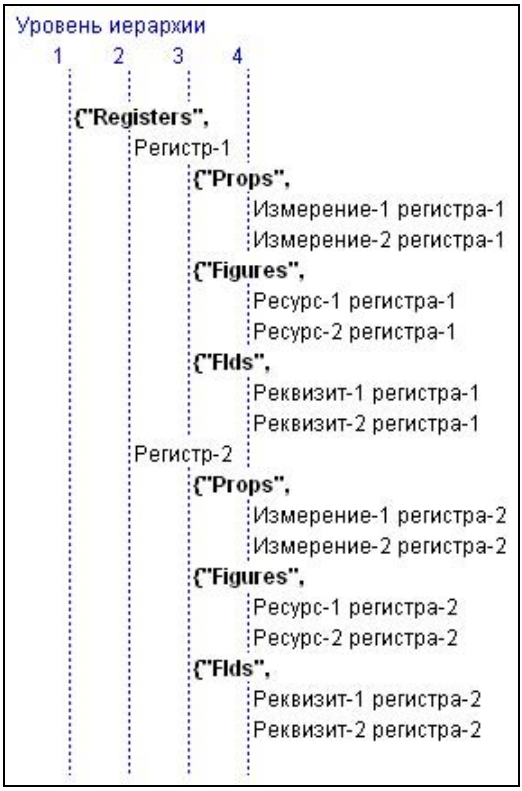



**Рис. 10.4.** Ветвь "дерева" от узла `{"Registers ",` (регистры)

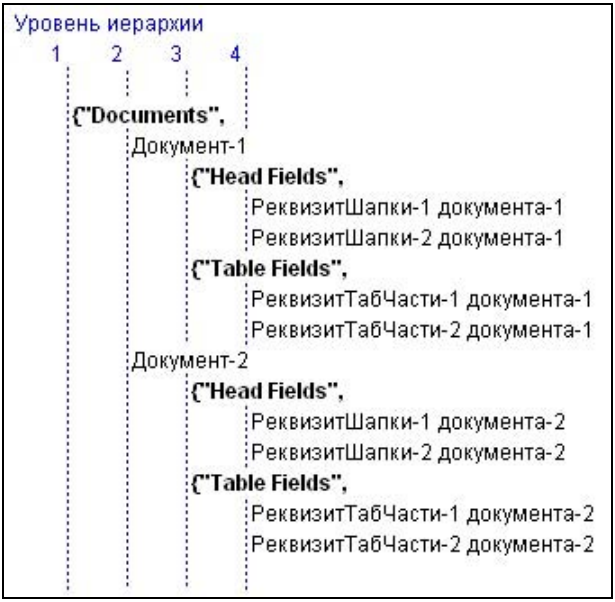



**Рис. 10.5.** Ветвь "дерева" от узла `{"Documents ",` (документы)

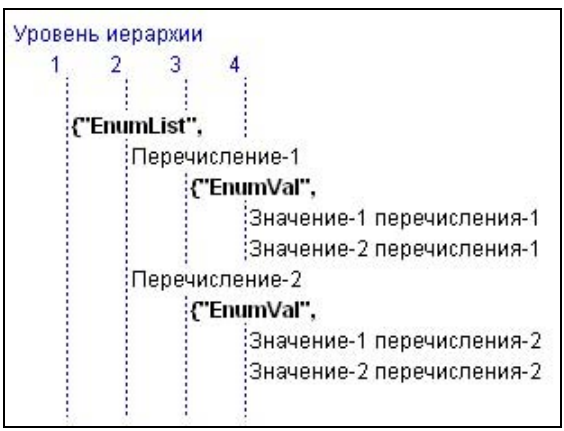



**Рис. 10.6.** Ветвь "дерева" от узла `{"EnumList ",` (перечисления)

Для преобразования значения системного идентификатора из десятичной системы счисления в 36-ричную можно использовать функцию `sp_Convert_Int10_to_Another` (листинг 10.7).

**Листинг 10.7. Перевод целого числа из десятичной системы счисления в другую с основанием [2 .. 36]**

```
function sp_Convert_Int10_to_Another(v10 : cardinal;
                                     BaseAnother : integer) : string;
// Перевод десятичного целого числа v10 в систему счисления
// с основанием BaseAnother.
// Функция возвращает представление числа
// в заданной системе счисления
Var
  vFrac : integer;  // Остаток по модулю (основание)
  vFracStr : string;// Строковое представление остатка
begin
 Result:='';
 if (BaseAnother>1) and (BaseAnother<=36) then begin
   repeat
    vFrac:=v10 MOD BaseAnother; // v10-(v10 div BaseAnother)*BaseAnother
    v10:=v10 DIV BaseAnother;   // trunc(v10/BaseAnother)
    if vFrac>9 then begin
      // Это буква латинского алфавита
      vFracStr:=CHR(ORD('A')-10+vFrac)
    end
    else begin
      // Переводим остаток в строковое представление
      STR(vFrac,vFracStr);
    end;
    Result:=Result+vFracStr;
   until v10=0;
   // Инвертируем результат
   Result:=sp_InversionString(Result);
 end;
end;
```

По сути, функция `sp_Convert_Int10_to_Another` — это аналог функции `_IdToStr` программы 1С:Предприятие, которую мы использовали в *разд. 4.4*.

Для обратной конвертации (из 36-ричного представления в десятичное) можно использовать функцию `sp_Convert_IntAnother_to_10` (листинг 10.8).

**Листинг 10.8. Перевод целого числа из системы счисления с основанием [2 .. 36] в десятичную**

```
function sp_Convert_IntAnother_to_10(vAnother : string;
                                     BaseAnother : integer) : cardinal;
// Перевод целого числа (vAnother) из системы счисления BaseAnother
// в десятичную систему.
// Функция возвращает представление числа
// в десятичной системе счисления.
var
  i : integer;
  Res, ValNum : integer;
begin
 Result:=0;
 if (BaseAnother>1) and (BaseAnother<=36) then begin
   vAnother:=Trim(vAnother);
   if length(vAnother)>0 then begin
     vAnother:=AnsiUpperCase(vAnother);
     for i:=1 to length(vAnother) do
     begin
      // Конвертируем текущий символ в число
      VAL(vAnother[i],ValNum,Res);
      if Res<>0 then begin
        // Если это не число, считаем, что это буква латинского алфавита,
        // и вычисляем по позиции
        ValNum:=10+ORD(vAnother[i])-ORD('A');
      end;
      // Вычисляем соответствующую степень и суммируем
      Result:= Result+sp_XpowerN(BaseAnother,length(vAnother)-i)*ValNum;
     end;
   end;
 end;
end;
```

В листинге 10.9 представлены тексты вспомогательных функций.

**Листинг 10.9. Вспомогательные функции для перевода числа из одной системы счисления в другую**

```
function sp_InversionString(Sx : string) : string;
// Функция инвертирует строку
Var
  i : integer;
begin
  Result:=Sx;
  if Length(Sx)>0 then begin
     Result:='';
     for i:=Length(Sx) downto 1 do
      begin
        Result:=Result+Sx[i];
     end;
  end;
end;

function sp_XpowerN(X, N :integer ):cardinal;
// X в степени N
var
 i : integer;
begin
  Result:=1;
  for i:=1 to N do
   begin
    Result:=Result*X;
  end;
end;
```

Примеры вызова:

- перевести число 20 из десятичной системы счисления в 36-ричную:
  `ShowMessage(sp_Convert_Int10_to_Another(20,36));`
- перевести число 2I из 36-ричной системы счисления в десятичную:
  `ShowMessage(IntToStr(sp_Convert_IntAnother_to_10('2I',36)));`

В листинге 10.10 представлен исходный текст функции, реализующей обработку информационного потока `Main MetaData Stream` из файла 1Cv7.MD.

**Листинг 10.10. Обработка информации потока `Main MetaData Stream` из файла 1Cv7.MD**

```
function sp_Extract_Info_from_1Cv7MD(List_1Cv7MD,
                                     ListINI : TStrings) : integer;
// Выделение нужной информации из файла 1Cv7_md_Main_MetaData_Stream.TXT:
// List_1Cv7MD - входной параметр
// (содержание файла 1Cv7_md_Main_MetaData_Stream.TXT),
// ListINI - результат (INI-структура)
begin
 Result:=0;
 if List_1Cv7MD<>NIL then begin
   if List_1Cv7MD.Count>0 then begin
     if ListINI<>NIL then begin
       // Преобразуем в "дерево"
       sp_Convert_Info_from_1Cv7MD_To_TreeText(List_1Cv7MD);
       // Выбираем константы
       sp_Actions_for_Consts(List_1Cv7MD, ListINI);
       // Обрабатываем раздел DocNumDef
       sp_Actions_for_DocNumDef(List_1Cv7MD, ListINI);
       // Выбираем общие реквизиты документов
       sp_Actions_for_GenJrnlFldDef(List_1Cv7MD, ListINI);
       // Выбираем и обрабатываем перечисления
       sp_Actions_for_Enums(List_1Cv7MD, ListINI);
       // Выбираем и обрабатываем справочники
       sp_Actions_for_SbCnts(List_1Cv7MD, ListINI);
       // Выбираем и обрабатываем документы
       sp_Actions_for_Documents(List_1Cv7MD, ListINI);
       // Выбираем и обрабатываем регистры
       sp_Actions_for_Registers(List_1Cv7MD, ListINI);
     end;
   end;
 end;
end;
```

Назначение функций:

- `sp_Convert_Info_from_1Cv7MD_To_TreeText` — конвертация информационного потока `Main MetaData Stream`;

- `sp_Actions_for_Consts` — обработка ветви дерева от узла `{"Consts",;`
- `sp_Actions_for_DocNumDef` — обработка ветви дерева от узла `{"DocNumDef",;`
- `sp_Actions_for_GenJrnlFldDef` — обработка ветви дерева от узла `{"GenJrnlFldDef",;`
- `sp_Actions_for_Enums` — обработка ветви дерева от узла `{"EnumList",;`
- `sp_Actions_for_SbCnts` — обработка ветви дерева от узла `{"SbCnts",;`
- `sp_Actions_for_Documents` — обработка ветви дерева от узла `{"Documents",;`
- `sp_Actions_for_Registers` — обработка ветви дерева от узла `{"Registers",.`

***Примечание***

Исходный текст функции `sp_Convert_Info_from_1Cv7MD_To_TreeText` представлен в листинге 10.5. Исходные тексты других функций здесь не приводтся по причине их "громоздкости", а также тривиальности в реализации вышеуказанных алгоритмов. Желающие могут посмотреть тексты этих функций в файлах исходных текстов Delphi-проекта в папке \Part_2\sp_1Cv7_Viewer_Tables.pas\006\ на сопроводительном компакт-диске книги.

В иллюстративных целях мы ограничились обработкой только некоторых узлов информационного потока `Main MetaData Stream`. В реальных условиях может возникнуть необходимость использования дополнительной информации из потока.

Здесь следует коснуться некоторых вопросов вычисления значений полей таблиц (DBF-файлов) как значений объектов 1С (с точки зрения их визуализации). Как отмечалось в *главе 9*, поля таблиц, имена которых начинаются с символов `SP`, являются реквизитами (измерениями, ресурсами) справочников, документов, регистров. Точно так же интерпретируется поле `VALUE` таблицы 1SCONST.DBF (значения констант и периодических реквизитов справочников). Кроме этого, в таблицах SC*.DBF (справочники) могут присутствовать поле `PARENTID` (соответствует "родителю" элемента справочника, если справочник иерархический) и поле `PARENTEXT` (соответствует владельцу элемента справочника, если у справочника есть владелец). Для всех этих полей при визуализации необходимо соответствующим образом вычислять значения.

Для вычисления значений полей таблиц (DBF-файлов), начинающихся с символов `SP`, может быть принят, например, такой алгоритм:

1. Из имени поля удаляются первые 2 символа. Оставшиеся символы — это системный идентификатор соответствующего объекта 1С.

2. По системному идентификатору из INI-файла (структура базы данных) считываются `Тип` и `Вид` объекта.
3. Если тип объекта — `Базовый`, то значение поля просто визуализируется "как есть".
4. Если тип объекта — `Агрегатный`, то значение вычисляется с учетом вида объекта.
5. Если тип объекта — `Неопределенный`, то значение вычисляется по специальным правилам.

Для вычисления значений полей, соответствующих агрегатному типу `Перечисление`, можно воспользоваться прямым считыванием параметра `Синоним` (из INI-файла) для заданного системного идентификатора (значения поля DBF-файла) с предварительной конвертацией значения поля из 36-ричной системы счисления в десятичную:

```
idSys:=IntToStr(sp_Convert_IntAnother_to_10(ЗначениеПоля,36));
SValueEnum := sp_GetFromINI_StringParam(fnINIstr,idSys,'Синоним','');
```

Для вычисления значений полей, соответствующих агрегатному типу `Справочник`, может быть принят такой алгоритм:

1. По системному идентификатору объекта (значение поля записи таблицы, преобразованное из 36-ричной системы счисления в десятичную) из INI-файла считывается параметр `Вид` (это системный идентификатор соответствующего справочника).
2. По значению параметра `Вид` вычисляется имя таблицы (DBF-файла):

   ```
   TableName := 'SC' + ЗначениеПараметраВид + '.dbf'
   ```

3. Проверяется, имеется ли в таблице `TableName` поле `DESCR` (имя элемента справочника). Для этого используется список полей таблицы (см. листинг 10.1).
4. Если поле `DESCR` существует, то формруется SQL-запрос для таблицы `TableName`:

   ```
   SELECT
      DESCR
   FROM
     'ИмяТаблицы или ПолноеИмяDBFФайла'
   WHERE
     ID='Значение поля'
   ```

В листинге 10.11 приведен текст функции `GetSbCntName`, возвращающей имя элемента справочника по его системному идентификатору.

**Листинг 10.11. Функция `GetSbCntName`, возвращающая имя элемента справочника**

```
function GetSbCntName(fnINI,idKind10,idSbCnt36 : string) : string;
// Получение имени элемента справочника:
// fnINI     - имя файла с INI-структурой,
// idKind10  - системный идентификатор (вид) справочника
//             (десятичная система счисления),
// idSbCnt36 - системный идентификатор элемента справочника
//             (36-ричная сист.счисления)
Var
 sFilter,  // Фильтр
 fnTable,  // Имя таблицы
 Sx : string;
begin
 Result:='';
 idKind10:=Trim(idKind10);
 if (length(Trim(idKind10))>0) and
    (length(Trim(idSbCnt36))>0) then begin
   // Проверяем наличие поля DESCR в таблице 'SC'+idKind10
   Sx:= sp_GetFromINI_StringParam(fnINI,'SC'+idKind10,'DESCR','0');
   if Trim(Sx) = '1' then begin
     // В этом справочнике есть поле DESCR
     //-----------------------
     // Вычисляем имя таблицы (DBF-файла) - справочника
     fnTable:='SC'+idKind10+'.dbf';
     fnTable:=ExtractFilePath(fnINI)+fnTable;
     fnTable:=#39+fnTable+#39;
     //-----------------------
     //-----------------------
     // Формируем фильтр
     sFilter:='ID=';
     sFilter:=sFilter+#39+idSbCnt36+#39;
     //-----------------------
     // Выбираем наименование элемента справочника
     // из таблицы(DBF-файла)
     Sx:=sp_GetFieldValue_TableDB('',fnTable,'DESCR',sFilter, '');
     Result:=Trim(Sx);
```

```
    end
    else begin
      // В этом справочнике НЕТ поля DESCR
      Result:='DESCR not found';
    end;
  end;
end;
```

В листинге 10.12 представлен исходный текст функции, возвращающей значение заданного поля заданной таблицы базы данных с помощью приведенного выше SQL-запроса.

**Листинг 10.12. Функция `GetFieldValue_from_TableDB`, возвращающая значение заданного поля заданной таблицы базы данных**

```
function GetFieldValue_from_TableDB(AliasName,
                                    TableName,
                                    FieldName,
                                    sFilter,
                                    sSort : string) : string;
// Получение значения заданного поля из заданной таблицы
// для заданного фильтра sFilter:
// AliasName - Alias из BDE-5 (может быть пустым),
// TableName - имя таблицы (может быть именем DBF-файла,
//             в этом случае заключается в кавычки),
// FieldName - имя поля, значение которого возвращается,
// sFilter   - фильтр (если по фильтру будет найдено несколько записей,
//             то функция возвращает значение поля первой записи),
// sSort     - список полей для сортировки (через запятую)
Var
 Q : TQuery;
 Yes : byte;
begin
 Result:='';
 TableName:=Trim(TableName);
 FieldName:=Trim(FieldName);
 sFilter:=Trim(sFilter);
 sSort:=Trim(sSort);
 AliasName:=Trim(AliasName);
```

```
 if Length(TableName)>0 then begin
   if Length(FieldName)>0 then begin
     Yes := 0;
     Q := TQuery.Create(Nil);
     TRY
      if Q<>NIL then begin
        Q.DatabaseName:=AliasName;
        Q.SQL.ADD('SELECT '+FieldName);
        Q.SQL.ADD('FROM ');
        Q.SQL.ADD(TableName);
        if length(sFilter)>0 then begin
          Q.SQL.ADD('WHERE ');
          Q.SQL.ADD(sFilter);
        end;
        if length(sSort)>0 then begin
          Q.SQL.ADD('ORDER BY ');
          Q.SQL.ADD(sSort);
        end;
        Q.Open;
        Q.First;
        Result:=Q.Fields[0].AsString;
        Yes := 1;
      end;
     FINALLY
      if Q<>NIL then begin
        Q.Close;
        if Yes<=0 then begin
           // Если попытка открыть набор данных завершилась ошибкой,
           // то выводим соответствующее сообщение
           ShowMessage('Err: SQL= '+Q.SQL.Text);
        end;
        Q.SQL.Clear;
        Q.Free;
        Q:=NIL;
      end;
     END;
   end;
 end;
end;
```

Пример вызова:

```
idParent:='     7';
TableName:=#39+'E:\1C_example.DB\SC12.DBF'+#39;
sFilter:='ID='+#39+idParent+#39;
sValue:=Trim(GetFieldValue_from_TableDB('',
                                        TableName,
                                        'DESCR',
                                        sFilter,
                                        ''));
```

Для вычисления значений полей, соответствующих агрегатному типу `Документ`, можно просто сформировать SQL-запрос к таблице 1SJOURN.DBF (журнал документов), фильтруя по полю `IDDOC` и прочитав значения полей `DOCNO` (номер документа) и `DATE` (дата документа). Вид документа (если нужно) можно вычислить, используя следующий алгоритм:

1. Считывается значение поля `DNPREFIX` из таблицы 1SJOURN.DBF.
2. Из значения поля `DNPREFIX` выделяются первые 10 символов (включая пробелы). Затем начальные и конечные пробелы удаляются. Это будет системный идентификатор вида документа (как объекта метаданных).
3. Из INI-файла для этого системного идентификатора считывается параметр `Синоним`.

В листинге 10.13 приведен текст функции `GetDocName`, возвращающей имя документа по его системному идентификатору.

**Листинг 10.13. Функция `GetDocName`, возвращающая имя документа по его системному идентификатору**

```
function Extract_KindDoc_from_DNPREFIX(DNPREFIX : string) : string;
// Выделение из префикса документа
// системного идентификатора его вида:
// DNPREFIX - значение поля DNPREFIX из журнала документов
begin
  Result:=DNPREFIX;
  System.Delete(Result,11,255);
end;

function GetDocName(fnINI,idDoc36 : string) : string;
// Получение имени документа из журнала документов:
// fnINI - имя файла с INI-структурой,
```

```
// idDoc36 - системный идентификатор документа
//           (36-ричная сист.счисления)
Var
  sFilter,
  idKind : string;
  fnTable : string;
  ListRes : TStrings;
begin
 Result:='';
 if length(Trim(idDoc36))>0 then begin
   if Trim(idDoc36)<>'0' then begin
     ListRes := TSTringList.Create;
     TRY
      //-----------------------
      // Название документа выбираем
      // из таблицы журнала документов
      fnTable:='1SJOURN'+'.dbf';
      fnTable:=ExtractFilePath(fnINI)+fnTable;
      fnTable:=#39+fnTable+#39;
      //-----------------------
      //-----------------------
      // Формируем фильтр
      sFilter:='IDDOC=';
      sFilter:=sFilter+#39+idDoc36+#39;
      //-----------------------
      // Читаем информацию из журнала документов
      sp_GetRec_TableDB('',fnTable,'*',sFilter, '',ListRes);
      // Выделяем код вида документа из префикса документа
      idKind:=ListRes.VALUES['DNPREFIX'];
      idKind:=Extract_KindDoc_from_DNPREFIX(idKind);
      //-----------------------
      // Формируем имя документа
      Result:='N';
      Result:=Result+' ' +Trim(ListRes.VALUES['DOCNO']);
      Result:=Result+' ['+Trim(ListRes.VALUES['DATE'])+']';
      // Читаем из INI-файла "Синоним" вида документа
      Result := Result
              + ' : '
```

```
              + Trim(sp_GetFromINI_StringParam(fnINI,
                                                Trim(idKind),
                                                'Синоним',''));
      //-----------------------
     FINALLY
      if ListRes<>NIL then begin
        ListRes.Clear;
        ListRes.Free;
        ListRes:=NIL;
      end;
     END;
   end;
 end;
end;
```

Текст функции `sp_GetRec_TableDB` приведен в листинге 10.14.

**Листинг 10.14. Функция, возвращающая значения заданных полей таблицы базы данных**

```
function sp_GetRec_TableDB(AliasName,TableName,FieldNames,
                           sFilter, sSort : string;
                           ListRes : TStrings) : integer;
// Получение списка значений заданных полей из заданной таблицы
// для заданного фильтра sFilter:
// AliasName  - Alias из BDE-5 (может быть пустым),
// TableName  - имя таблицы (может быть именем DBF-файла,
//              в этом случае оно заключается в кавычки),
// FieldNames - имена полей,
// sFilter    - фильтр (если в наборе данных больше, чем одна
//              запись,то значения полей возвращаются из первой),
// sSort      - список полей для сортировки,
// ListRes    - здесь возвращаются значения полей записи таблицы.
// Функция возвращает размер списка ListRes
Var
  Q   : TQuery;
  Yes : byte;
  i   : integer;
begin
```

```
Result:=0;
if ListRes<>NIL then begin
  ListRes.Clear;
  TableName:=Trim(TableName);
  FieldNames:=Trim(FieldNames);
  if length(FieldNames)<=0 then FieldNames:='*';
  sFilter:=Trim(sFilter);
  sSort:=Trim(sSort);
  AliasName:=Trim(AliasName);
  if Length(TableName)>0 then begin
    Yes := 0;
    Q := TQuery.Create(Nil);
    TRY
     if Q<>NIL then begin
       Q.DatabaseName:=AliasName;
       Q.SQL.ADD('SELECT '+FieldNames);
       Q.SQL.ADD('FROM ');
       Q.SQL.ADD(TableName);
       if length(sFilter)>0 then begin
         Q.SQL.ADD('WHERE ');
         Q.SQL.ADD(sFilter);
       end;
       if length(sSort)>0 then begin
         Q.SQL.ADD('ORDER BY ');
         Q.SQL.ADD(sSort);
       end;
       Q.Open;
       // Позиционируемся на первой строке открытого набора данных
       Q.First;
       //-----------------------------
       // Читаем значения полей записи открытого набора данных
       for i:=0 to (Q.Fields.Count-1) do
        begin
         ListRes.ADD(Q.Fields[i].FieldName+'='+Q.Fields[i].AsString);
       end;
       //-----------------------------
       Result:=ListRes.Count;
       Yes := 1;
```

```
      end;
     FINALLY
       if Q<>NIL then begin
         Q.Close;
         if Yes<=0 then begin
            // Если попытка открыть набор данных завершилась ошибкой,
            // то выводим соответствующее сообщение
            ShowMessage('Err: SQL= '+Q.SQL.Text);
         end;
         Q.SQL.Clear;
         Q.Free;
         Q:=NIL;
       end;
     END;
   end;
 end;
end;
```

Для вычисления значений поля `PARENTID` таблиц (DBF-файлов), если справочник иерархический, может быть принят, например, такой алгоритм:

1. Проверяется, существует ли в *этой* же таблице (значение поля которой мы вычисляем) поле `DESCR` (наименование элементов справочника):

   ```
   Sx:= sp_GetFromINI_StringParam(fnINIstr,TableName,'DESCR','0');
   if Sx= '1' then begin
      // Поле DESCR существует
   end;
   ```

2. Если поле `DESCR` существует, то формруется SQL-запрос для *этой* же таблицы:

   ```
   SELECT
      DESCR
   FROM
     'ИмяЭтойТаблицы или ПолноеИмяDBFФайла'
   WHERE
     ID='Значение поля PARENTID'
   ```

На рис. 10.7 представлен результат визуализации записи файла SC12.DBF (что соответствует элементу справочника `ФизЛица`).

На рис. 10.8 представлен результат визуализации записи файла RA78.DBF (регистр `УчетВыплатФизЛицам`).

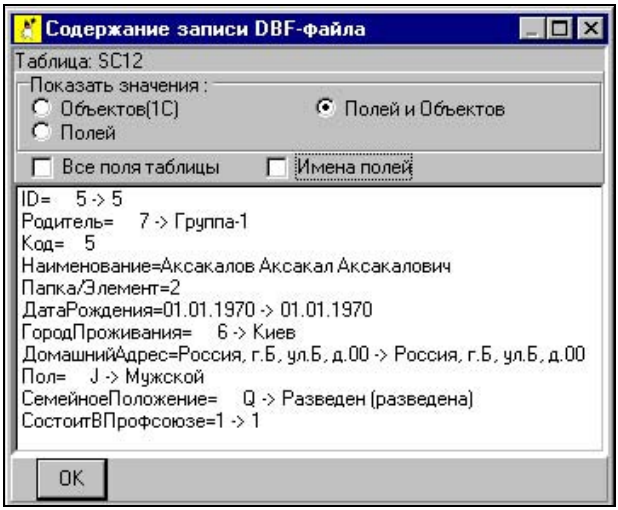



**Рис. 10.7.** Визуализация записи файла SC12.DBF (справочник `ФизЛица`)

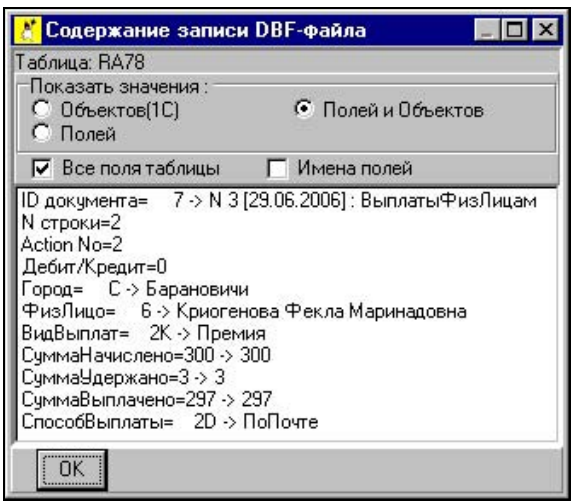



**Рис. 10.8.** Визуализация записи файла RA78.DBF (регистр `УчетВыплатФизЛицам`)

Для вычисления значений поля `PARENTEXT` таблиц (DBF-файлов), если у справочника есть владелец, может быть принят, например, такой алгоритм:

1. Из имени этой же таблицы (для которой вычисляется значение поля `PARENTEXT`) выделяется системный идентификатор соответствующего объекта (справочника).

2. По системному идентификатору справочника из INI-файла (структура базы данных) считывается значение параметра `Владелец` объекта (там хранится системный идентификатор владельца для данного справочника).
3. По системному идентификатору владельца вычисляется имя таблицы-владельца (DBF-файла):

```
tnOwner:='SC'+'СистемныйИдентификаторВладельца'+'.dbf'
```

4. Проверяется, существует ли в таблице (у владельца) поле `DESCR` (имя элемента справочника-владельца):

```
Sx:= sp_GetFromINI_StringParam(fnINIstr, tnOwner,'DESCR','0');
if Sx='1' then begin
   // Поле DESCR существует
end;
```

5. Если поле `DESCR` существует, то формруется SQL-запрос для таблицы `tnOwner`:

```
SELECT
   DESCR
FROM
  'ИмяТаблицыВладельца'
WHERE
  ID='Значение поля PARENTEXT'
```

На рис. 10.9 представлен результат визуализации записи файла SC35.DBF (поле `Владелец`).

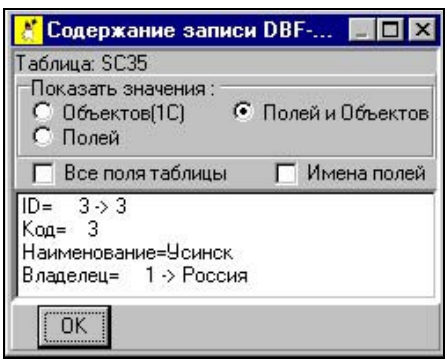



**Рис. 10.9.** Визуализация записи файла SC35.DBF (справочник `ГородаСтранМира`)

Для периодических реквизитов можно использовать следующий алгоритм:

1. По системному идентификатору заданного периодического реквизита вычисляется его представление в 36-ричной системе счисления:

   ```
   idSys36:=sp_Convert_Int10_to_Another(sp_StrToInt(idSys,0),36);
   ```

2. Значение переменной `idSys36` дополняется слева нужным количеством пробелов (в нашем случае должно быть всего 4 символа).
3. Считывается значение поля `ID` таблицы (справочника) из записи, соответствующей элементу `idObj36` справочника, для которого фильтруются значения периодического реквизита.
4. Значения периодического реквизита `idSys36` для значения поля `ID` выбираются следующим образом:

   ```
   SELECT
     VALUE
   FROM
     'ПолныйПутьКФайлу\1SCONST.dbf'
   WHERE
     OBJID='idObj36'
   AND
     ID='idSys36'
   ```

На рис. 10.10 представлен результат визуализации записи файла 1SCONST.DBF (значение периодического реквизита справочника).

Для вычисления значений объектов неопределенного типа (а также значений типа `Справочник` неопределенного вида и `Документ` неопределенного вида) придется применять дополнительные меры.

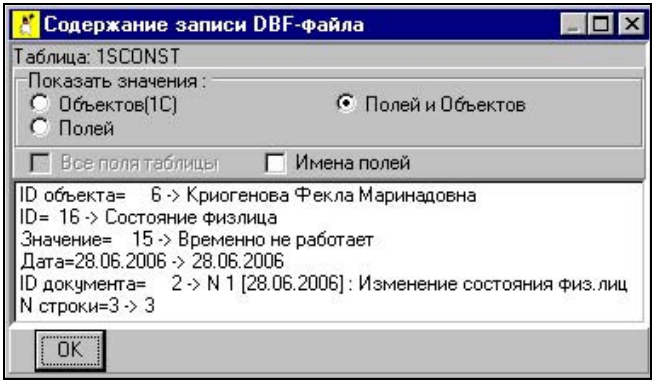



**Рис. 10.10.** Визуализация записи файла 1SCONST.DBF (значение периодического реквизита справочника)

В табл. 10.12 приведены примеры сравнения того, как значения заданы в контексте 1С:Предприятие и как они отражены в DBF-файлах.

***Таблица 10.12.*** *Реквизиты неопределенного типа и/или вида*

| **Тип реквизита** | **Тип (вид) значения** | **Значение в контексте 1С** | **Значение поля DBF-файла** |
|---|---|---|---|
| Неопределенный | N 10.2 | 999.01 | N 999.01 |
| Неопределенный | S | Тип S длина=22 END | STип S длина=22 END |
| Неопределенный | D | 24.09.2006 | D20060924 |
| Неопределенный | Справочник.ФизЛица | Элемент справочника с кодом 6, системный идентификатор 6 | B1 C 6 |
| Неопределенный | Неопределенное | Нет значения | U |
| Неопределенный | Документ. ВыплатыФизЛицам | Номер 3 от 29.06.2006, системный идентификатор 7 | O1 2G 7 |
| Неопределенный | Перечисление. ВидВыплатФизЛицам | МатПомощь | E1 2H 2L |
| Справочник неопределенного вида | ГородаСтранМира | Элемент справочника с кодом 3, системный идентификатор 3 | Z 3 |
| Справочник неопределенного вида | Неопределенный | Нет значения | 0 0 |
| Документ неопределенного вида | ВыплатыФизЛицам | Номер 1 от 14.07.2006, системный идентификатор 5 | 2G 5 |
| Документ неопределенного вида | Неопределенный | Нет значения | 0 0 |

Небольшой анализ позволяет сделать следующие выводы.

Для реквизитов неопределенного типа:

- первый символ в значении поля обозначает тип объекта (`N` — число, `S` — строка, `B` — справочник, `O` — документ, `E` — перечисление, `D` — дата, `U` — неопределенный и т. д.);
- если посмотреть формат поля (файл 1Cv7.DD), то видно, что его длина равна 23, тип — строка (независимо от хранящихся данных): это значит (как отмечалось в *разд. 4.7.1*), что максимальная длина значения реквизита, которое можно записать в это поле, 22 символа (независимо от типа данных);
- значения всех объектов базового типа (кроме даты) можно визуализировать явно (то, что прочитали из поля), удалив первый символ; для объектов типа `Дата` нужно дополнительно конвертировать дату из числового формата *YYYYMMDD* в ее строковый эквивалент *ДД.ММ.ГГГГ*;
- символ `U` в начале строки — это отсутствие значения.

Для всех объектов агрегатного типа:

- второе слово в значении (разделитель слов — символ пробела) — это системный идентификатор вида объекта в 36-ричной системе счисления;
- третье слово в значении — это системный идентификатор соответствующей ссылки в 36-ричной системе счисления (для справочника — это значение поля `ID` соответствующей таблицы; для документа — значение поля `IDDOC` соответствующей таблицы; для перечисления — системный идентификатор соответствующего значения).

Для реквизитов типа `Справочник` или `Документ` неопределенного вида:

- если посмотреть формат поля (файл 1Cv7.DD), то видно, что его длина равна 13, тип — строка (независимо от хранящихся данных);
- первое слово в значении поля (разделитель слов — символ пробела) — это системный идентификатор вида объекта в 36-ричной системе счисления;
- второе слово в значении — это системный идентификатор соответствующей ссылки в 36-ричной системе счисления (для справочника — это значение поля `ID` соответствующей таблицы; для документа — значение поля `IDDOC` соответствующей таблицы);
- если оба слова `= "0"`, то это отсутствие значения (строка символов `" 0 0"`).

Все это справедливо и для периодических реквизитов (таблица `1SCONST`).

В принципе, с учетом вышеизложенного алгоритм вычисления значений для реквизитов неопределенного типа (вида) напрашивается сам собой и весьма

навязчиво. Думаю, что не следует останавливаться здесь на этом подробнее (кто желает — может посмотреть исходные тексты на сопроводительном компакт-диске книги).

В таблице `1SCONST` есть поле `TVALUE` (рис. 10.11).

| OBJID | ID | DATE | PARTNO | VALUE | DOCID | TIME | ACTNO | LINENO | TVALUE |
|---|---|---|---|---|---|---|---|---|---|
| 2TZ | 4KB | 31.12.2002 | 0 | 1B | 0 | 0 | 0 | 0 | |
| 2TZ | 4KC | 31.12.2002 | 0 | E1 5J8 LT9 | 0 | 0 | 0 | 0 | 1 |
| 2TZ | 4KD | 31.12.2002 | 0 | B1 4O8 32 | 0 | 0 | 0 | 0 | 1 |
| 2TZ | 4KE | 31.12.2002 | 0 | B1 4KT A8 | 0 | 0 | 0 | 0 | 1 |
| 2TZ | 4KI | 30.12.1899 | 0 | 6.70 | 0 | 0 | 0 | 0 | |
| 2TZ | 4KI | 31.08.2000 | 0 | 6.70 | 0 | 0 | 0 | 0 | |
| 2TZ | 4KI | 01.05.2001 | 0 | 6.70 | 0 | 0 | 0 | 0 | |
| 2TZ | 4KN | 01.01.2002 | 0 | 1.0 | 0 | 0 | 0 | 0 | |

**Рис. 10.11.** Поле `TVALUE` таблицы `1SCONST`

Это поле принимает значение `"1"` в тех случаях, когда периодический реквизит неопределенного типа (по метаданным) принимает определенное значение агрегатного типа (поле `VALUE`). Если реквизиту назначен агрегатный тип, но не назначен вид, то значение поля `TVALUE = "0"`. В остальных случаях (из наблюдений) поле содержит три пробела.

Аналогично, для каждого реквизита (поля, начинающиеся с символов `"SP"`) неопределенного типа (в справочниках, документах) в таблице присутствует поле, имеющее такое же имя, но с символом `"T"` в начале имени поля.

Пример 1:

```
------------------------------------
F=SP139      |(P)Реквизит_Неопреде|C    |23     |0
F=TSP139     |                    |C    |3      |0
------------------------------------
```

Пример 2 (рис. 10.12):

```
------------------------------------
F=SP2857     |(P)ВидСтатус        |C    |23     |0
F=TSP2857    |                    |C    |3      |0
------------------------------------
```

На рис. 10.13 представлен результат визуализации записи файла SC2543 (справочник `ДвиженияРегистров`).

На рис. 10.14 представлен результат визуализации значения периодического реквизита неопределенного типа (файл 1SCONST.DBF).

| SP2538 | SP2539 | SP2857 | TSP2857 | SP2541 |
|---|---|---|---|---|
| ПартииОтданные | - | E1 98 2TA | 1 | 1AU |
| ПартииНаличие | + | E1 98 2TD | 1 | 1AU |
| ПартииОтданные | - | E1 98 2TD | 1 | 1AU |
| | | U | | 0 |
| ПартииНаличие | + | E1 98 2TB | 1 | 150 |
| ПартииНаличие | + | E1 98 2TD | 1 | 150 |
| Покупатели | - | E1 B0 B1 | 1 | 150 |

**Рис. 10.12.** Реквизит неопределенного типа `ВидСтатус` справочника `ДвиженияРегистров`

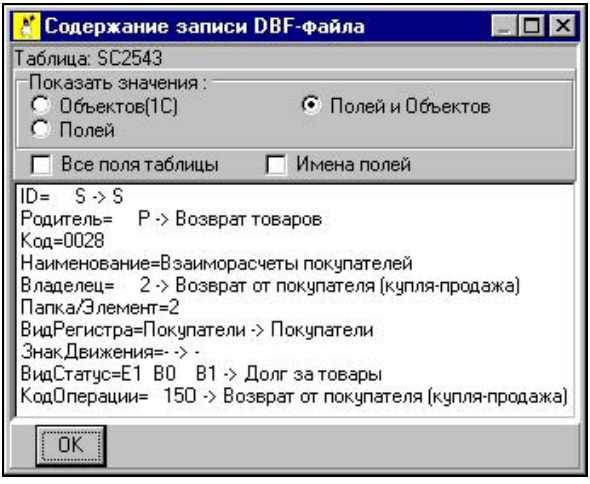



**Рис. 10.13.** Визуализация записи файла SC2543.DBF (справочник `ДвиженияРегистров`). Реквизит неопределенного типа

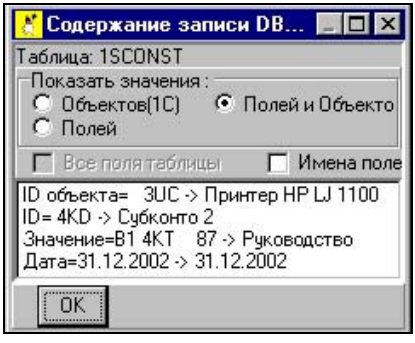



**Рис. 10.14.** Визуализация значения периодического реквизита неопределенного типа

Как мы увидели в *части II*, при разработке универсального программного обеспечения (для доступа к базам данных 1С без программы 1С:Предприятие) существенно возрастают "накладные расходы" по созданию инструментария (библиотек), ориентированных на "расшифровку" информационной структуры базы данных, с целью поддержки объектной модели хранения информации, принятой в системе 1С:Предприятие. В общем случае это может стать существенным ограничением для использования механизма SQL-запросов к базам данных 1С. То есть при выборе способа обмена информацией с 1С (OLE Automation или SQL-запросы) нужно следовать здравому смыслу в каждом конкретном случае и выбирать механизм доступа, наиболее эффективный для решения прикладной задачи.

Совсем другое дело, если нужно разработать программное обеспечение, ориентированное на узкую задачу и конкретную информационную структуру базы данных 1С. В этом случае можно создать эффективный инструментарий достаточно быстро и просто. В следующей главе мы рассмотрим именно такой случай на конкретном примере.

***Примечание***

Напомню, что представленные примеры исходных текстов — это упрощенная иллюстрация к рассматриваемым вопросам. Назначение этих примеров — дать представление о возможном подходе при решении задач информационного обмена с системой 1С:Предприятие. При работе с реальными базами 1С может потребоваться их дальнейшая доработка.

# Глава 11

# Остатки ТМЦ на складах. Пример

Товарно-материальные ценности (ТМЦ) — это часть большой группы активов (материально-производственных запасов или МПЗ) предприятия. В процессе хозяйственной деятельности предприятия его работники используют предметы труда и средства труда. Предметы труда (материалы) — это сырье, топливо, материалы, тара, запасные части, инвентарь, материальные ценности, полученные от выбытия основных средств, и т. д. ТМЦ на предприятии группируются в систематизированный номенклатурный справочник, где каждому виду материалов соответствует отдельный номенклатурный номер. Номенклатурный номер используется в дальнейшем для отражения наличия конкретного материала и его движения на предприятии. Поступление ТМЦ на предприятие может происходить разными способами (приобретение, безвозмездное получение, изготовление, разборка ликвидированных основных средств и т. д.). Для хранения ТМЦ на предприятии создаются должным образом оборудованные склады, кладовые или специально приспособленные площадки. Хранение ТМЦ на складах организуется таким образом, чтобы была обеспечена возможность их быстрой приемки, отпуска и проверки наличия. В целом, учет ТМЦ на предприятии (в частности, хранение) — это очень сложный и трудоемкий процесс, реализованный в комплексной конфигурации 1С:Предприятие определенным образом. В этой главе мы на конкретном примере рассмотрим вопросы получения информации о движении и остатках ТМЦ на складах с использованием SQL-запросов к базе данных 1С в простейшей реализации.

В контексте комплексной конфигурации 1С:Предприятие учет остатков ТМЦ на складах предприятия организован с использованием регистра `ОстаткиТМЦ`. На рис. 11.1 представлена структура этого регистра.

Попробуем создать простой инструментарий, который позволил бы получить остаток (а также движение за период) ТМЦ на заданную дату в разрезе склада, с применением механизма SQL-запросов к базе данных 1С.

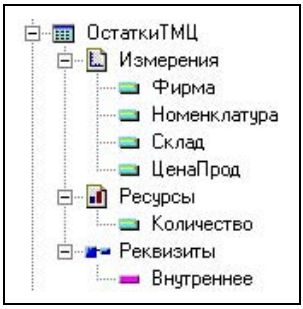



**Рис. 11.1.** Структура регистра ОстаткиТМЦ

***Примечание***

Не думаю, что в данном случае нужно оговаривать такие нюансы, как "получить ли остаток *на* дату или *по* дату", поскольку это важно с точки зрения бухучета, но не программной реализации инструментария.

Физически регистр ОстаткиТМЦ организован в виде двух таблиц (листинг 11.1):

- RG405 — движение по регистру ОстаткиТМЦ;
- RA405 — остатки на регистре ОстаткиТМЦ на конец каждого периода.

**Листинг 11.1. Структура таблиц RG405 и RA405 (регистр Остатки ТМЦ)**

```
#==TABLE no 473     : Регистр (Дв.) ОстаткиТМЦ
# Name     |Descr                    |Type[A/S/U]|DBTableName|ReUsable
T=RA405    |Регистр (Дв.) ОстаткиТМЦ      |A          |RA405      |1
#-----Fields-------
# Name       |Descr                |Type|Length|Precision
F=IDDOC      |ID Document's        |C   |9     |0
F=LINENO     |LineNo               |N   |4     |0
F=ACTNO      |Action No            |N   |6     |0
F=DEBKRED    |Flag Debet/Kredit    |N   |1     |0
F=SP4062     |(P)Фирма             |C   |9     |0
F=SP408      |(P)Номенклатура      |C   |9     |0
F=SP418      |(P)Склад             |C   |9     |0
F=SP3117     |(P)ЦенаПрод          |N   |16    |2
F=SP411      |(P)Количество        |N   |16    |5
```

```
F=SP420      |(P)Внутреннее        |N   |2     |0

#==TABLE no 472     : Регистр ОстаткиТМЦ
# Name     |Descr                  |Type[A/S/U]|DBTableName|ReUsable
T=RG405    |Регистр ОстаткиТМЦ            |A          |RG405      |1
#-----Fields-------
# Name       |Descr                |Type|Length|Precision
F=PERIOD     |Period Registr       |D   |8     |0
F=SP4062     |(P)Фирма             |C   |9     |0
F=SP408      |(P)Номенклатура      |C   |9     |0
F=SP418      |(P)Склад             |C   |9     |0
F=SP3117     |(P)ЦенаПрод          |N   |16    |2
F=SP411      |(P)Количество        |N   |16    |5
```

Здесь значение поля `PERIOD` в таблице `RG405` соответствует дате (остатки рассчитываются на конец месяца), на которую рассчитаны остатки ТМЦ по данному регистру (в разрезе всех измерений). Поскольку среди измерений регистра (см. рис. 11.1) есть объекты `Номенклатура` (поле `SP408`) и `Склад` (поле `SP418`), это гарантирует наличие уже рассчитанных остатков на дату (заданную в поле `PERIOD`) в разрезе номенклатуры (ТМЦ) и склада. Величина остатка хранится в объекте `Количество` (поле `SP411`).

Казалось бы, можно создать простейший SQL-запрос к таблице `RG405`, где в качестве фильтра указать дату, системные идентификаторы соответствующей номенклатуры, склада — и задача решена. Если дата, на которую мы желаем получить остаток ТМЦ по складу, совпадает с одной из дат, на которую уже рассчитаны остатки, то это так и есть. Задачу позволит решить SQL-запрос, представленный в листинге 11.2.

**Листинг 11.2. SQL-запрос к таблице `RG405`**

```
SELECT
  SP411
FROM
  "ПолныйПуть_к_DBF-файлу\RG405.dbf"
WHERE
  PERIOD = "ЗаданнаяДата"
AND
  SP408 = "ТМЦ (СистемныйИдентификатор)"
AND
  SP418 = "Склад (СистемныйИдентификатор)"
```

Но если дата, на которую нужно получить остаток ТМЦ на складе, не совпадает с какой-либо расчетной датой (на которую остатки уже рассчитаны), то такой простой подход, естественно, не применим. В этом случае нужно получить остаток на "ближайшую" расчетную дату, которая меньше заданной (листинг 11.2), а затем проанализировать движение по регистру (таблица `RA405`) и вычислить суммарный `Приход` и `Расход` ТМЦ (в разрезе номенклатуры и склада) за период от "ближайшей" расчетной даты (конец месяца) до заданной даты. Очевидно, что остаток на заданную дату будет равен:

```
ОстатокНаЗаданДату = ОстатокНаРасчДату + Приход — Расход
```

Здесь следует отметить важный нюанс. Дело в том, что в таблице `RG405` все даты (поле `PERIOD`) имеют формат ***01**.ММ.ГГГГ* (рис. 11.2).

| PERIOD | SP4062 | SP408 | SP418 | SP3117 | SP411 |
|---|---|---|---|---|---|
| 01.01.2004 | 1 | 3BSH | 5H | 0 | 1 |
| 01.02.2004 | 1 | 3BSH | 5H | 0 | 1 |
| 01.03.2004 | 1 | 3BSH | 5H | 0 | 1 |
| 01.04.2004 | 1 | 3BSH | 5H | 0 | 1 |
| 01.05.2004 | 1 | 3BSH | 5H | 0 | 1 |
| 01.06.2004 | 1 | 3BSH | 5H | 0 | 1 |
| 01.07.2004 | 1 | 3BSH | 5H | 0 | 1 |
| 01.08.2004 | 1 | 3BSH | 5H | 0 | 1 |
| 01.09.2004 | 1 | 3BSH | 5H | 0 | 1 |
| 01.10.2004 | 1 | 3BSH | 5H | 0 | 1 |
| 01.11.2004 | 1 | 3BSH | 5H | 0 | 1 |
| 01.12.2004 | 1 | 3BSH | 5H | 0 | 1 |

**Рис. 11.2.** Содержание таблицы `RG405` (`ОстаткиТМЦ`)

Казалось бы, можно предположить, что все остатки рассчитаны на начало месяца. Но это не так — с точностью до наоборот. Все остатки рассчитаны на конец месяца, а значения поля `PERIOD` соответствуют дате *конца* месяца. А это значит, что при расчете остатков (в таблице `RG405`) нужно следовать двум правилам:

1. Если дата (например, 31.05.2004), на которую мы желаем получить остаток ТМЦ, соответствует дате конца месяца, то в SQL-запросе нужно ее модифицировать таким образом, чтобы значение дня даты было равно 01 (в данном случае 01.05.2004).
2. Если дата (например, 12.05.2004), на которую мы желаем получить остаток ТМЦ, *не* соответствует дате конца месяца, то в SQL-запросе нужно ее модифицировать таким образом, чтобы значение дня даты было равно 01, а значение месяца — предыдущему месяцу (в данном случае 01.04.2004).

В общем случае для получения остатка ТМЦ на заданном складе и на заданную дату можно применить следующий алгоритм:

1. Пользователь выбирает:
   - дату, на которую нужно получить остаток (`DateRest`);
   - ТМЦ (номенклатуру), остаток по которой нужно получить (`idMatSys`);
   - склад, по которому нужно получить остаток (`idStoreSys`).
2. Программа анализирует дату (`DateRest`), введенную пользователем, и определяет ее новое значение (`DateRestX`) по следующим правилам:
   - если дата (например, 31.05.2004), на которую пользователь желает получить остаток ТМЦ, соответствует дате конца месяца, то в SQL-запросе ее значение модифицируется таким образом, чтобы значение дня даты было равно 01 (в данном случае 01.05.2004);
   - если дата (например, 12.05.2004), на которую пользователь желает получить остаток ТМЦ, не соответствует дате конца месяца, то в SQL-запросе ее значение модифицируется таким образом, чтобы значение дня даты было равно 01, а значение месяца — предыдущему месяцу (в данном случае 01.04.2004).

   ***Примечание***

   Можно, конечно, модифицировать только значение месяца, но тогда придется учитывать переход с марта на февраль. Например, 30 марта — это не конец месяца, но 30 февраля в природе не бывает (во всяком случае в настоящее время).

3. Программа ищет "ближайшую" расчетную дату (`DateRestCalc`), которая меньше или равна `DateRestX`:

```
SELECT
   MAX(PERIOD)
   FROM
     "ПолныйПуть_к_DBF-файлу\RG405.dbf"
   WHERE
     SP408 = 'idMatSys'
   AND
     SP418 = 'idStoreSys'
   AND
     PERIOD<='DateRestX'
```

4. Если дата `DateRestCalc` существует, выбирается остаток на эту дату:

```
SELECT
  SP411
```

```
FROM
  "ПолныйПуть_к_DBF-файлу\RG405.dbf"
WHERE
  SP408 = 'idMatSys'
AND
  SP418 = 'idStoreSys'
AND
  PERIOD='DateRestCalc'
```

5. Если дата `DateRestCalc` *не* существует, то принимаем, что остаток `RestOnDateCalc` на дату `DateRestCalc` = 0.
6. Если `DateRestX` = `DateRestCalc`, то задача считается решенной и остаток на дату `DateRest` составляет: `RestOnDate` = `RestOnDateCalc`.
7. Если `DateRestX > DateRestCalc`, то нужно:
   - найденную дату `DateRestCalc` (например, 01.04.2004) конвертировать в соответствующую дату конца месяца `DateRestCalcX` (в данном случае 30.04.2004);
   - отобрать движение ТМЦ по складу за период `[DateRestCalcX ... DateRest]` (*исключая* дату `DateRestCalcX`) и рассчитать суммарные `Приход` (`Valplus`) и `Расход` (`Valminus`) за этот период;
   - вычислить остаток ТМЦ по складу на дату `DateRest`:

     `"RestOnDate" = "RestOnDateCalc" + "Valplus" - "Valminus"`

На этом будем считать задачу решенной. В принципе, ничего сложного в реализации этого алгоритма нет. Но есть существенный нюанс. Если посмотреть на структуру таблицы `RA405`, то видно, что в ее составе нет поля, которое бы содержало дату движения по регистру, но есть поле `IDDOC` (системный идентификатор документа, к которому "привязано" это движение). Отсюда следует, что значения даты и времени, к которым приурочена любая строка таблицы `RA405`, можно получить из журнала документов `1SJOURN` *(см. предыдущие главы)*.

В листинге 11.3 представлен текст SQL-запроса на получение списка документов, формирующих движение по регистру `ОстаткиТМЦ` за период.

**Листинг 11.3. SQL-запрос к таблице `1SJOURN` на получение списка документов**

```
SELECT
  IDDOC
FROM
  "ПолныйПуть_к_DBF-файлу\1SJOURN.dbf" J
```

```
WHERE
  RF405>0
AND
  J."DATE" > 'DateRestCalcX'
AND
  J."DATE" <= 'DateRest';
ORDER BY
  IDDOC
```

Таким образом, для решения нашей задачи нужно "задействовать" 5 таблиц из базы данных 1С:

- RG405 — остатки на регистре ОстаткиТМЦ на конец каждого периода. Поля:
  - PERIOD — дата, на которую рассчитаны остатки;
  - SP408 — номенклатура (системный идентификатор);
  - SP418 — склад (системный идентификатор);
  - SP411 — количество (остаток ТМЦ);
- RA405 — движение по регистру ОстаткиТМЦ. Поля:
  - IDDOC — документ (системный идентификатор);
  - DEBKRED — признак Приход/Расход;
  - SP408 — номенклатура (системный идентификатор);
  - SP418 — склад (системный идентификатор);
  - SP411 — количество (остаток ТМЦ);
- 1SJOURN — журналы документов. Поля:
  - IDDOC — системный идентификатор документа;
  - DATE — дата документа;
  - RF405 — признак того, что этот документ может принимать участие в формировании записей таблицы RG405;
- SC55 — справочник Склады. Поля:
  - ID — системный идентификатор элемента справочника;
  - CODE — значение свойства Код элемента справочника;
  - DESCR — значение свойства Наименование элемента справочника;
- SC84 — справочник Номенклатура. Поля:
  - ID — системный идентификатор элемента справочника;

- `CODE` — значение свойства `Код` элемента справочника;
- `DESCR` — значение свойства `Наименование` элемента справочника.

На рис. 11.3 представлена схема, иллюстрирующая связи между указанными таблицами.

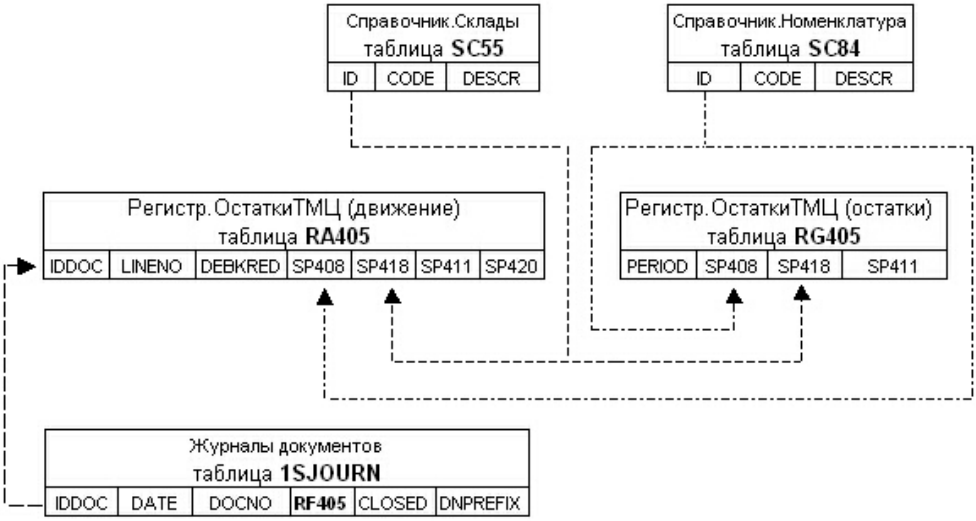



**Рис. 11.3.** Схема, иллюстрирующая связи между таблицами, участвующими в формировании движения по регистру `ОстаткиТМЦ`

В конечном итоге для решения этой узкой задачи нам не потребуется проводить анализ информационной структуры базы данных, что существенно ускорит и упростит программирование.

### *Примечание*

Здесь следует отметить, что все примеры данной главы будут тестироваться (по ходу изложения) на базе данных комлексной конфигурации. Из этого следует, что:

- те, кто желает самостоятельно поэкспериментировать, должны иметь доступ к комплексной конфигурации;
- естественно, что результат у каждого экспериментатора будет свой.

Создадим новый Delphi-проект. На рис. 11.4 представлена форма проекта.

### *Примечание*

Исходные тексты находятся на сопроводительном компакт-диске книги в папке \Part_2\Chapter_11\sp_1Cv7_GetRest.PAS\01\.

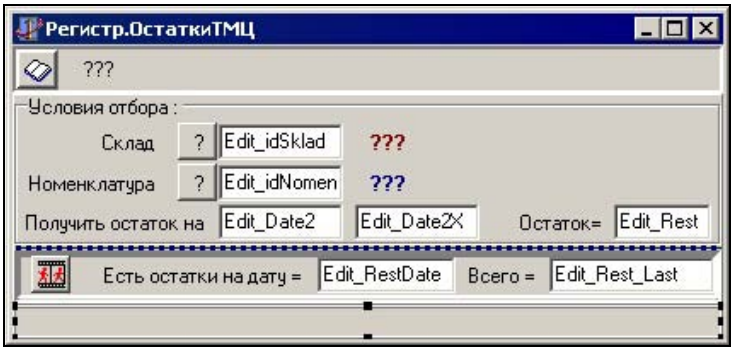



**Рис. 11.4.** Форма проекта sp_1Cv7_GetRest.DPR

Элементы управления формы:

- **Склад**:
  - кнопка **?** — выбор склада;
  - текстовое поле (`Edit_idSklad`) — системный идентификатор выбранного склада;
- группа **Номенклатура**:
  - кнопка **?** — выбор ТМЦ;
  - текстовое поле (`Edit_idNomenkl`) — системный идентификатор номенклатуры;
- группа **Получить остаток на**:
  - текстовое поле (`Edit_Date2`) — дата, на которую желаем получить остаток;
  - текстовое поле (`Edit_Date2X`) — модифицированное значение из поля `Edit_Date2` (для выборки по таблице `RG405`);
- текстовое поле **Остаток** (`Edit_Rest`) — величина остатка на желаемую дату;
- кнопка "бегущие человечки" — запуск алгоритма получения остатка на желаемую дату;
- текстовое поле **Есть остатки на дату** (`Edit_RestDateEnd`) — "ближайшая" расчетная дата (на которую рассчитаны остатки), значение которой меньше или равно желаемой дате;
- текстовое поле **Всего** (`Edit_Rest_Last`) — остаток ТМЦ на "ближайшую" расчетную дату.

Для выбора склада из таблицы SC55 и номенклатуры из таблицы SC84 создадим диалоговую форму (рис. 11.5)

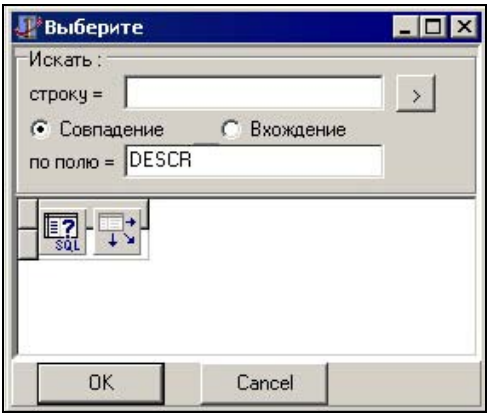



**Рис. 11.5.** Форма для выбора склада и номенклатуры для фильтрации по таблицам RA405 и RG405

Не вдаваясь в детали исходного кода формы (см. рис. 11.5), отметим только, что список складов для выбора в интерактивном режиме можно получить, сформировав SQL-запрос (листинг 11.4).

**Листинг 11.4. SQL-запрос к таблице SC55 на получение списка складов**

```
SELECT
  CODE,
  DESCR,
  ID
FROM
  "ПолныйПуть_к_DBF-файлу\SC55.dbf"
WHERE
  ISFOLDER<>1
ORDER BY
  DESCR
```

Номенклатурный список (ТМЦ) для выбора в интерактивном режиме можно получить, сформировав SQL-запрос (листинг 11.5).

**Листинг 11.5. SQL-запрос к таблице SC55 на получение номенклатурного списка (ТМЦ)**

```
SELECT
  CODE,
  DESCR,
  ID
FROM
  "ПолныйПуть_к_DBF-файлу\SC84.dbf"
WHERE
  ISFOLDER<>1
ORDER BY
  DESCR
```

В листинге 11.6 представлен фрагмент обработчика события "нажатие кнопки "бегущие человечки"" (запуск алгоритма вычисления остатка).

**Листинг 11.6. Фрагмент обработчика нажатия кнопки "бегущие человечки"**

```
procedure TForm1.SpeedButton6Click(Sender: TObject);
Var
  Sx,
  sFilter,
  fn,          // Имя поля таблицы
  tn : string; // Имя таблицы
begin
...
// Модифицируем дату (на которую желаем получить остаток)
// для использования ее в SQL-запросе к таблице RG405
 Edit_Date2X.Text:=ReCalcDate_for_SelectRest(Edit_Date2.Text);
//-----------------------------------------
// Определим "ближайшую" расчетную дату остатка
sFilter:='';
// Склад
sFilter:=sFilter+'SP418='+#39+Edit_idSklad.Text+#39;
sFilter:=sFilter+' and ';
// Номенклатура
sFilter:=sFilter+'SP408='+#39+Edit_idNomenkl.Text+#39;
sFilter:=sFilter+' and ';
```

```
// Ограничение по дате
sFilter:=sFilter+'PERIOD <= '+#39+Trim(Edit_Date2X.Text)+#39;
tn:='"'+DirDB+'RG405'+'.dbf'+'"';
// Выбираем максимальное значение поля для заданного фильтра
fn:='MAX(PERIOD)';
// Формируем и выполняем SQL-запрос
Sx:=sp_GetFieldValue_TableDB('',tn,fn,sFilter,'');
Edit_RestDateEnd.Text:=Trim(Sx);
//-----------------------------------------
if length(Edit_RestDateEnd.Text)>0 then begin
// Такая дата существует
  //-----------------------------------------
  // Читаем остаток за найденную "ближайшую" дату
  sFilter:='';
  // Склад
  sFilter:=sFilter+'SP418='+#39+Edit_idSklad.Text+#39;
  sFilter:=sFilter+' and ';
  // Номенклатура
  sFilter:=sFilter+'SP408='+#39+Edit_idNomenkl.Text+#39;
  sFilter:=sFilter+' and ';
  // "Ближайшая" дата остатка
  sFilter:=sFilter+' PERIOD = '+#39+Trim(Edit_RestDateEnd.Text)+#39;
  // Выбираем количество (остаток ТМЦ) для заданного фильтра
  fn:='SP411';
  // Формируем и выполняем SQL-запрос
  Sx:=sp_GetFieldValue_TableDB('',tn,fn,sFilter,'');
  Edit_Rest_Last.Text:=Trim(Sx);
  //-----------------------------------------
end
else begin
  ShowMessage('Остатков на эту дату не обнаружено');
end;
...
end;
```

Думаю, что здесь комментарии излишни. Функция `sp_GetFieldValue_TableDB` была подробно рассмотрена в предыдущей главе, а функция `ReCalcDate_for_SelectRest` никакого интереса с технической точки зрения не представляет (реализацию этих функций можно увидеть в исходных текстах проекта).

Скомпилируем наш Delphi-проект и запустим его на выполнение. Выберем склад и ТМЦ (рис. 11.6).

На рис. 11.7 представлен результат выполнения обработчика (листинг 11.6).

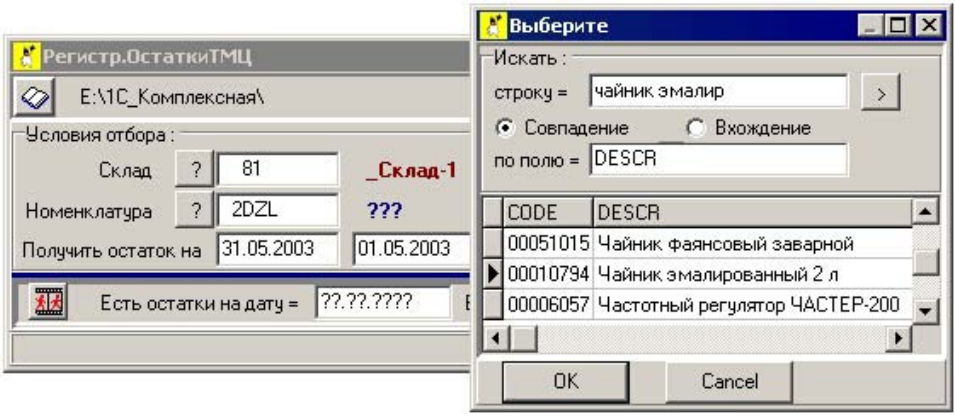



**Рис. 11.6.** Выбор ТМЦ в справочнике `Номенклатура`

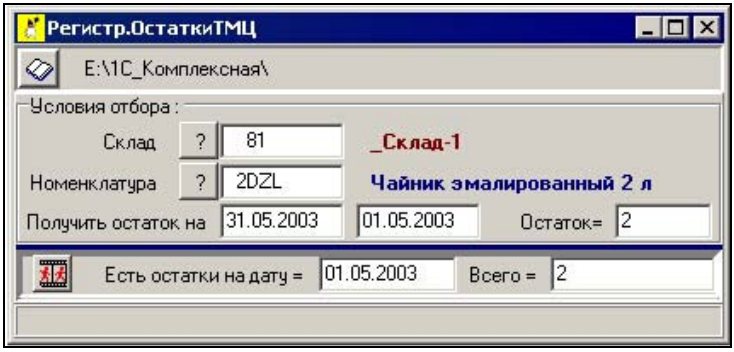



**Рис. 11.7.** Остаток ТМЦ по заданному складу (SQL-запрос)

Теперь запустим программу 1С:Предприятие, откроем эту же конфигурацию, выберем стандартный отчет "Остатки ТМЦ" и "запустим" отчет для этого же склада, номенклатуры и на эту же дату. Результат представлен на рис. 11.8.

Как видим, результаты одинаковы (по величине остатка). Но по времени выполнения наша программа (SQL-запросы) существенно быстрее (ради чего мы все это и затеяли). Для сравнения: у меня (на локальной базе данных) остаток был получен:

- в программе sp_1Cv7_GetRest.EXE — 3–4 с;
- в программе 1С:Предприятие — 50 с.

**Остатки ТМЦ на складах**

На дату: 31.05.03

По всем фирмам. По всем юр. лицам. По всем упр. аналитикам. По номенклатуре "Чайник эмалированный 2 л". По складу "_Склад-1".

| ТМЦ | Ед. | Усред. себест-ть с НДС (руб.) | _Склад-1 Кол - во |
|---|---|---|---|
| **Материалы и оборудование вспомогательные** | | | |
| **Посуда** | | | |
| **Посуда стальная** | | | |
| Чайник эмалированный 2 л | шт | 285.15 | 2.000 |

**Рис. 11.8.** Остаток ТМЦ по заданному складу (стандартный отчет 1С:Предприятие "Остатки ТМЦ")

Модернизируем наш Delphi-проект. Посмотрим, каким образом можно получить выборку документов, участвующих в формировании движения по регистру `ОстаткиТМЦ` за период дат от "ближайшей" расчетной даты до даты, на которую мы желаем получить остаток ТМЦ. На рис. 11.9 представлена форма проекта.

### *Примечание*

Исходные тексты этого примера находятся на сопроводительном компакт-диске книги в папке \Part_2\Chapter_11\sp_1Cv7_GetRest.PAS\02\.

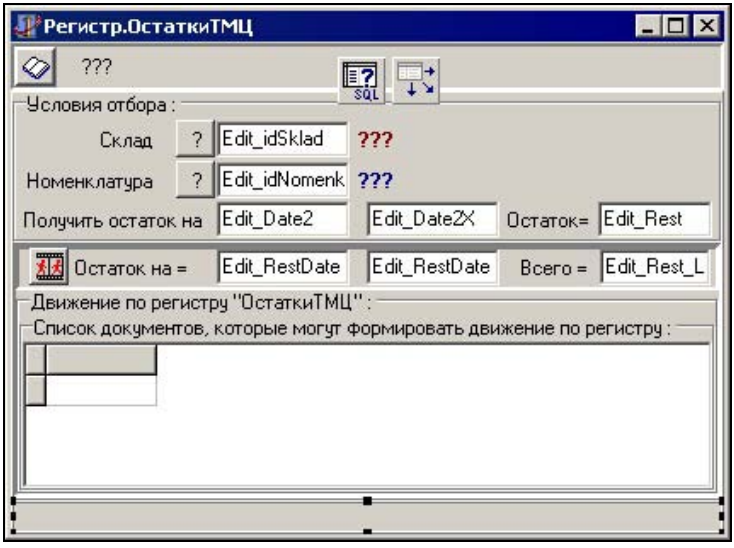



**Рис. 11.9.** Форма проекта sp_1Cv7_GetRest.DPR

В листинге 11.7 представлен текст функции, формирующей список документов.

**Листинг 11.7. Функция, формирующая список документов**

```
function TForm1.OpenListDoc : integer;
begin
  Result:=0;
  QueryDoc.Close;
  QueryDoc.SQL.Clear;
  QueryDoc.SQL.ADD('SELECT ');
  QueryDoc.SQL.ADD('IDDOC, ');
  QueryDoc.SQL.ADD('J."DATE", ');
  QueryDoc.SQL.ADD('DOCNO, ');
  QueryDoc.SQL.ADD('DNPREFIX ');
  QueryDoc.SQL.ADD('FROM ');
  QueryDoc.SQL.ADD('"'+DirDB+'1SJOURN'+'.dbf'+'" J');
  QueryDoc.SQL.ADD('WHERE ');
  // Отбираем только те документы, которые могут формировать
  // движение по регистру "ОстаткиТМЦ"
  QueryDoc.SQL.ADD('RF405>0 ');
  QueryDoc.SQL.ADD('AND ');
  // Дата начала периода - начало следующего месяца
  // (после даты остатков)
  QueryDoc.SQL.ADD('J."DATE" > '+#39+Edit_RestDateEndX.Text+#39);
  QueryDoc.SQL.ADD('AND ');
  // Дата конца периода - заданная дата
  QueryDoc.SQL.ADD('J."DATE" <= '+#39+Edit_Date2.Text+#39);
  QueryDoc.SQL.ADD('ORDER BY ');
  // Сортируем по дате и системному идентификатору документов
  QueryDoc.SQL.ADD('J."DATE",IDDOC ');
  QueryDoc.Open;
  Result:=QueryDoc.RecordCount;
end;
```

Текст функции прост и не нуждается в пояснениях.

В листинге 11.8 представлен фрагмент обработчика (см. листинг 11.6) с учетом открытия списка документов.

**Листинг 11.8. Доработанный обработчик нажатия кнопки "бегущие человечки"**

```
procedure TForm1.SpeedButton6Click(Sender: TObject);
Var
 ...
begin
  ...
  fn:='SP411';
  Sx:=sp_GetFieldValue_TableDB('',tn,fn,sFilter,'');
  Edit_Rest_Last.Text:=Trim(Sx);
  // Вычисляем дату конца месяца, который соответствует
  // найденной дате расчета остатков
  Edit_RestDateEndX.Text:=
                 sp_ConvertThisDateToEndMonth(Edit_RestDateEnd.Text);
  // Если даты не совпадают, формируем список документов
  if Trim(Edit_RestDateEndX.Text)<>Trim(Edit_Date2.Text) then begin
     // Формируем список документов за период от даты последнего
     // расчета остатков (конец месяца) до заданной даты
     OpenListDoc;
  end;
  ...
end;
```

Скомпилируем проект, запустим его на выполнение и протестируем соответствующим образом. На рис. 11.10 представлен результат формирования списка документов.

Поле **Имя документа** в таблице (рис. 11.10) — вычисляемое. Его значение определяется в обработчике события `OnCalcFields` компонента `QueryDoc` нашего проекта. Для того чтобы визуализировать наименование документов (вернее, вид документов), в каталоге базы данных должен присутствовать файл 1Cv7_str.INI (мы подробно рассматривали это в предыдущих главах).

Здесь следует отметить, что в списке присутствуют только те документы, которые участвовали в формировании движения по регистру `ОстаткиТМЦ` за заданный период времени. Но далеко не все они участвовали в формировании движения именно этого ТМЦ по заданному складу.

Продолжим модернизацию нашего Delphi-проекта. Откроем таблицу `RA405` (движение по регистру `ОстаткиТМЦ`) и отфильтруем записи по заданному складу и номенклатуре (листинг 11.9).

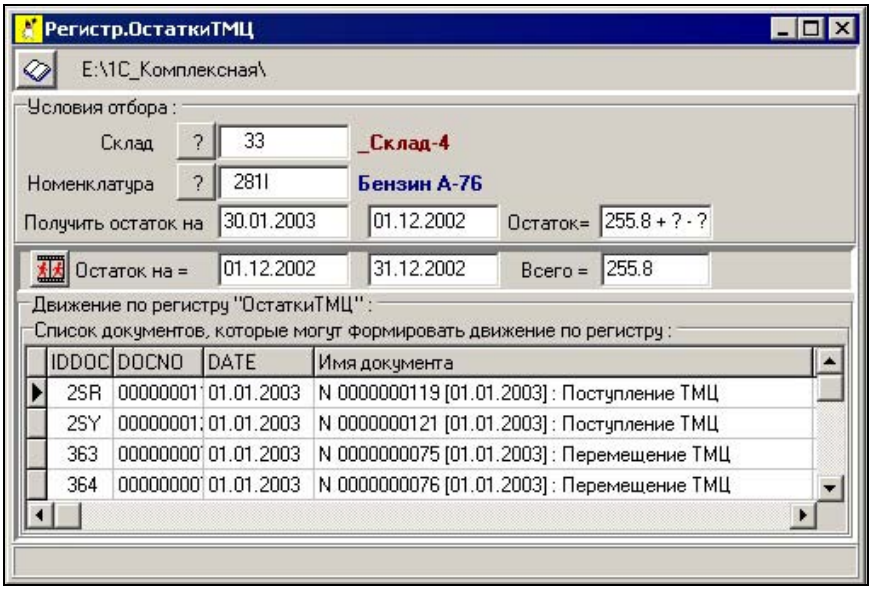



**Рис. 11.10.** Результат формирования списка документов

**Листинг 11.9. Открытие и фильтрация таблицы `RA405` (движение по регистру `ОстаткиТМЦ`)**

```
function TForm1.OpenListReg(NrWhat : integer) : integer;
// Открытие движения по регистру "ОстаткиТМЦ"
begin
  Result:=0;
  QueryReg.Close;
  QueryReg.SQL.Clear;
  QueryReg.SQL.ADD('SELECT ');
  QueryReg.SQL.ADD('IDDOC, ');
  QueryReg.SQL.ADD('LINENO, ');
  QueryReg.SQL.ADD('DEBKRED, ');
  QueryReg.SQL.ADD('SP411, ');
  QueryReg.SQL.ADD('SP420 ');
  QueryReg.SQL.ADD('FROM ');
  QueryReg.SQL.ADD('"'+DirDB+'RA405'+'.dbf'+'"');
  QueryReg.SQL.ADD('WHERE ');
  QueryReg.SQL.ADD('SP418='+#39+Edit_idSklad.Text+#39);
  QueryReg.SQL.ADD('AND ');
```

```
  QueryReg.SQL.ADD('SP408='+#39+Edit_idNomenkl.Text+#39);
  if NrWhat>=0 then begin
     QueryReg.SQL.ADD('AND ');
     if NrWhat=0 then begin
        QueryReg.SQL.ADD('DEBKRED<=0 ');
     end
     else begin
        QueryReg.SQL.ADD('DEBKRED>0 ');
     end;
  end;
  QueryReg.Open;
  Result:=QueryReg.RecordCount;
end;
```

Здесь, как видим, присутствует *все* движение по заданным номенклатуре и складу, независимо от даты. А это нам не подходит. Как видно из схемы на рис. 11.3, обе таблицы — `1SJOURN` (журнал документов) и `RA405` (движение по регистру `ОстаткиТМЦ`) — имеют общее поле `IDDOC`. Следовательно, мы можем сделать дополнительную фильтрацию по набору данных (таблица `RA405`) с учетом списка документов (таблица `1SJOURN`) за заданный период. Для этого есть, как минимум, два способа.

*Первый* способ заключается в том, что можно использовать *вложенный SQL-подзапрос* (листинг 11.10).

**Листинг 11.10. Сложный SQL-запрос по таблице `RA405`**

```
SELECT
  IDDOC,
  LINENO,
  DEBKRED,
  SP411,
  SP420
FROM
  "ПолныйПуть_к_DBF-файлу\RA405.dbf"
WHERE
  SP418='Edit_idSklad.Text'
AND
  SP408='Edit_idNomenkl.Text'
AND
```

```
  IDDOC IN (
            SELECT
              IDDOC
            FROM
              "ПолныйПуть_к_DBF-файлу\1SJOURN.dbf" J
            WHERE
              RF405>0
            AND
              J."DATE" >  'Edit_RestDateEndX.Text'
            AND
              J."DATE" <= 'Edit_Date2.Text'
           )
```

*Второй способ* — использовать обработчик `OnFilterRecord` Delphi-компонента `TQuery` (листинг 11.11).

**Листинг 11.11. Обработчик `OnFilterRecord` компонента `TQuery` (Delphi)**

```
procedure TForm1.QueryRegFilterRecord(DataSet: TDataSet;
  var Accept: Boolean);
Var
  idDoc : string;
begin
  Accept:=FALSE;
  if QueryDoc.Active then begin
     idDoc:=QueryReg.FieldByName('IDDOC').AsString;
     if QueryDoc.Locate('IDDOC',idDoc,[]) then Accept:=TRUE;
  end;
end;
```

Первый способ должен обеспечить (в случае хранения базы данных на сервере MS SQL Server) более быстрое выполнение, поскольку все операции по фильтрации данных выполняются на сервере.

> ***Примечание***
>
> На сопроводительном компакт-диске книги в папке \Part_2\Chapter_11\sp_1Cv7_GetRest.PAS\03\ приведен пример, где реализован первый способ фильтрации.

На рис. 11.11 представлена форма проекта.

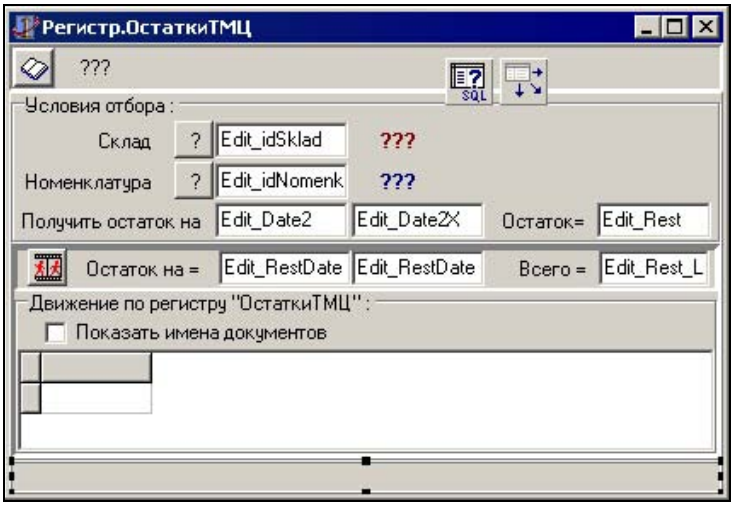



**Рис. 11.11.** Форма проекта sp_1Cv7_GetRest.dpr

На рис. 11.12 представлен результат фильтрации записей по таблице `RA405` (первый способ).

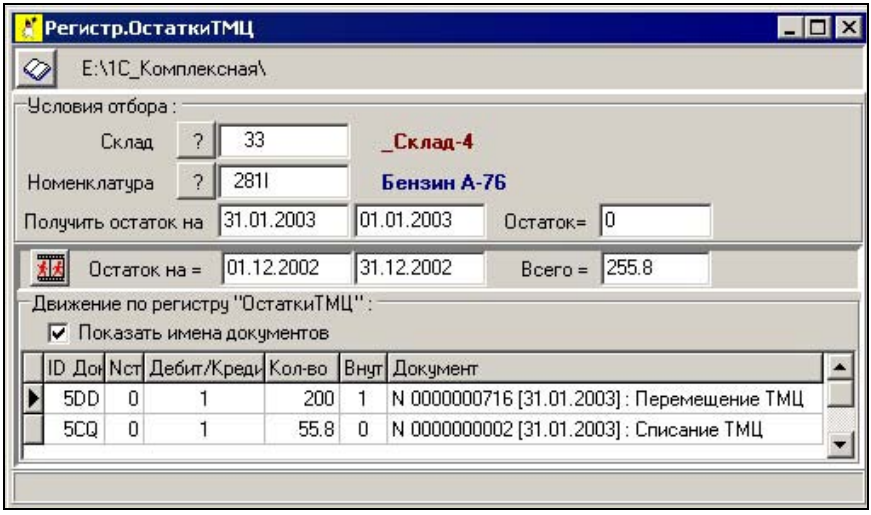



**Рис. 11.12.** Результат фильтрации записей о движении по регистру `ОстаткиТМЦ`

Для того чтобы визуализировать наименование документов (вернее вид документов), в каталоге базы данных должен присутствовать файл 1Cv7_str.INI

(см. предыдущий раздел). В случае отсутствия этого файла в имени документа будет отсутствовать вид документов (рис. 11.13).

| ID Док. | Nстр | Дебит/Кредит | Кол-во | Внутр. | Документ |
|---|---|---|---|---|---|
| 5DD | 0 | 1 | 200 | 1 | N 0000000716 [31.01.2003] : |
| 5CQ | 0 | 1 | 55.8 | 0 | N 0000000002 [31.01.2003] : |

**Рис. 11.13.** Результат фильтрации записей о движении по регистру ОстаткиТМЦ. Визуализация наименований документов без файла 1Cv7_str.INI

После того как отфильтрованы записи движения по регистру, можно рассчитать остаток ТМЦ на заданную дату (листинг 11.12).

**Листинг 11.12. Расчет остатка ТМЦ на заданную дату**

```
...
// Остаток на "ближайшую" расчетную дату
Rest0:=StrToFloat(Edit_Rest_Last.Text);
...
Rplus  := 0;
Rminus := 0;
// Сканируем записи движения по регистру
QueryReg.First;
while not QueryReg.EOF do
 begin
  if QueryReg.FieldByName('DEBKRED').AsInteger<=0 then begin
     // Это приход ТМЦ
     Rplus := Rplus + QueryRegSP411.AsFloat;
  end
  else begin
     // Это расход ТМЦ
     Rminus := Rminus + QueryRegSP411.AsFloat;
  end;
  Application.ProcessMessages;
  QueryReg.Next;
end;
// Вычисляем остаток на заданную дату
RestX := Rest0 + Rplus - Rminus;
...
```

***Примечание***

В папке \Part_2\Chapter_11\sp_1Cv7_GetRest.PAS\04\ на сопроводительном компакт-диске книги приведен пример, в котором реализован второй способ фильтрации записей по таблице `RA405` (с использованием обработчика `OnFilterRecord` компонента `TQuery`). Здесь мы этот пример рассматривать не будем.

В приведенных примерах кроме остатка ТМЦ на складе мы получили "расшифровку" его движения. Если в этом нет необходимости, то программу можно еще упростить. Вместо одного SQL-запроса (см. листинг 11.10) можно выполнить два запроса — получить приход и расход ТМЦ по складу за период (листинги 11.13 и 11.14), и затем рассчитать остаток на дату (листинг 11.15).

**Листинг 11.13. Получение прихода ТМЦ по складу за период**

```
SELECT
  SUM(SP411)
FROM
  "ПолныйПуть_к_DBF-файлу\RA405.dbf"
WHERE
  SP418='Edit_idSklad.Text'
AND
  SP408='Edit_idNomenkl.Text'
AND
  DEBKRED <= 0
AND
  IDDOC IN (
            SELECT
              IDDOC
            FROM
              "ПолныйПуть_к_DBF-файлу\1SJOURN.dbf" J
            WHERE
              RF405>0
            AND
              J."DATE" >  'Edit_RestDateEndX.Text'
            AND
              J."DATE" <= 'Edit_Date2.Text'
           )
```

**Листинг 11.14. Получение расхода ТМЦ по складу за период**

```
SELECT
  SUM(SP411)
FROM
  "ПолныйПуть_к_DBF-файлу\RA405.dbf"
WHERE
  SP418='Edit_idSklad.Text'
AND
  SP408='Edit_idNomenkl.Text'
AND
  DEBKRED > 0
AND
  IDDOC IN (
            SELECT
              IDDOC
            FROM
              "ПолныйПуть_к_DBF-файлу\1SJOURN.dbf" J
            WHERE
              RF405>0
            AND
              J."DATE" >  'Edit_RestDateEndX.Text'
            AND
              J."DATE" <= 'Edit_Date2.Text'
           )
```

**Листинг 11.15. Расчет остатка ТМЦ на заданную дату**

```
...
Rrest0:=StrToFloat(Edit_Rest_Last.Text);
...
// Приход ТМЦ на склад
OpenListReg(0);
QueryReg.First;
Rplus := QueryReg.Fields[0].AsFloat;
// Расход ТМЦ на складе
OpenListReg(1);
QueryReg.First;
Rminus := QueryReg.Fields[0].AsFloat;
// Расчет остатка на дату
RrestX := Rrest0 + Rplus - Rminus;
...
```

***Примечание***

В папке \Part_2\Chapter_11\sp_1Cv7_GetRest.PAS\05\ на сопроводительном компакт-диске книги приведен пример, реализующий этот способ вычисления остатка ТМЦ.

На рис. 11.14 представлен результат вычисления остатка ТМЦ по заданному складу.

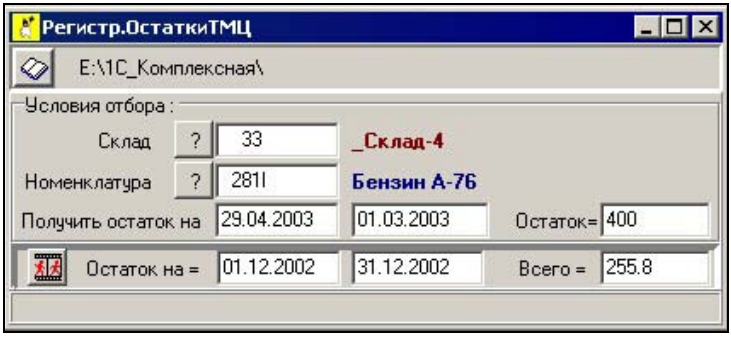



**Рис. 11.14.** Результат расчета остатка ТМЦ на складе

В этой главе мы рассмотрели расчет остатков заданного ТМЦ на заданном складе. Если требуется найти, например, остаток заданного ТМЦ в целом по предприятию, то (при формировании SQL-запросов) из фильтра исключается условие органичения выборки по складу (поле `SP418`), например, так:

```
...
QueryReg.SQL.ADD('WHERE ');
if length(Trim(Edit_idSklad.Text))>0 then begin
   QueryReg.SQL.ADD('SP418='+#39+Edit_idSklad.Text+#39);
   QueryReg.SQL.ADD('AND ');
end;
QueryReg.SQL.ADD('SP408='+#39+Edit_idNomenkl.Text+#39);
QueryReg.SQL.ADD('AND ');
QueryReg.SQL.ADD('IDDOC IN (...)');
...
```

***Примечание***

Напомню, что приведенные примеры исходных текстов — это упрощенная иллюстрация к рассматриваемым вопросам. Назначение этих примеров — дать представление о возможном подходе при решении задач информационного обмена с системой 1С:Предприятие. При работе с реальными базами 1С может потребоваться их дальнейшая доработка.

## Часть III

# Программа 1С:Предприятие в среде Windows

Программа 1С:Предприятие, будучи "обычным" приложением, функционирующим в среде Windows, использует системное окружение, включая соответствующие ресурсы (в частности, системный реестр Windows), которые предоставляет операционная система. В *части I* мы уже рассматривали вопрос визуализации окна программы 1С:Предприятие с применением стандартных средств управления процессами в среде операционной системы Windows. Посмотрим, что еще можно "выжать" полезного из программы 1С:Предприятие как приложения в среде Windows, используя возможности операционной системы.

# Глава 12

# Список зарегистрированных баз даннных системы 1С:Предприятие

В предыдущих главах мы (когда нужно было найти каталог базы данных 1С) "напрягали" потенциального пользователя, вынуждая его вручную отыскивать расположение базы данных 1С. Но если запустить программу 1С:Предприятие (или Конфигуратор 1С), то на экране появляется диалоговое окно выбора нужной базы данных 1С из списка зарегистрированных (рис. 12.1).

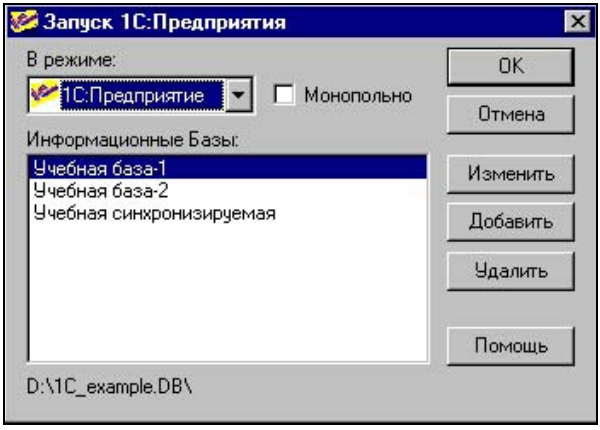



**Рис. 12.1.** Список конфигураций (баз данных) 1С, открытый в программе 1С:Предприятие

Следовательно, программа 1С:Предприятие где-то хранит список своих баз данных (конфигураций). Если запустить редактор системного реестра Windows (программа RegEdit.EXE) и найти раздел (для Windows 2000/XP):

```
HKEY_CURRENT_USER\Software\1C\1CV7\7.7\Titles
```

то видно (рис. 12.2), что в этом разделе системного реестра программа 1С:Предприятие формирует список своих баз данных.

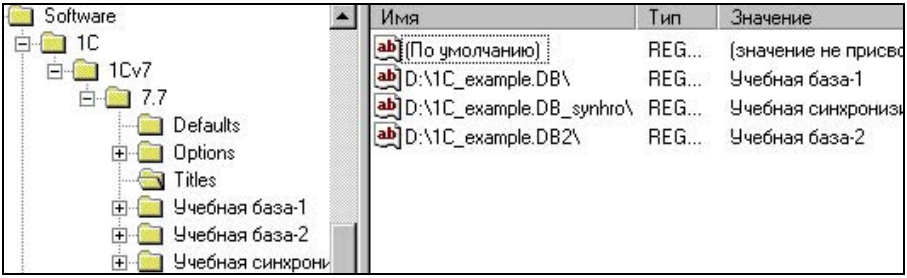



**Рис. 12.2.** Список конфигураций (баз данных) 1С в системном реестре Windows

В листинге 12.1 представлен исходный текст функции, считывающей список баз данных 1С из системного реестра Windows.

**Листинг 12.1. Функция чтения списка баз данных 1С из системного реестра**

```
function Load_ListDB_from_Registry(KeyName : string;
                                   List : TStrings;
                                   YesDetect_1Cv7md : boolean) : integer;
// Чтение списка баз данных 1С из системного реестра:
// KeyName - полное имя раздела системного реестра,
// откуда читаем информацию. Значения для Windows-2000/XP:
//   KeyName= Software\1C\1CV7\7.7\Titles
//   RootKey= HKey_Current_User
// List - здесь возвращаем список баз данных 1С,
// YesDetect_1Cv7md - если TRUE, то учитываем наличие файла 1Cv7.md.
var
  sNameDB,       // Имя конфигурации (базы данных) 1С
  fnMD : string; // Полное имя файла 1Cv7.MD
  Reg  : TRegistry;
  i    : integer;
  Yes  : byte;
begin
 Result:=0;
 if List<>NIL then begin
   List.Clear;
   KeyName:=Trim(KeyName);
```

```
if length(KeyName)>0 then begin
  Reg:=TRegistry.Create;
  try
   Reg.RootKey:= HKey_Current_User;
   if Reg.OpenKey(KeyName,False) then begin
     // Читаем полный путь к конфигурациям
     // (базам данных) 1С
     Reg.GetValueNames(List);
     if List.Count>0 then begin
       //--------------------------------
       //Формируем список баз данных 1С
       i:=-1;
       while i<(List.Count-1) do
       begin
        i:=i+1;
        Yes  := 0;
        if YesDetect_1Cv7md then begin
          // Проверка существования файла 1Cv7.md
          // для данной конфигурации.
          // Вычисляем полное имя MD-файла конфигурации
          fnMD := List[I]+'1cv7.md';
          // Проверяем, существует ли MD-файл
          if not FileExists(fnMD) then begin
             // MD-файл НЕ существует, удаляем из списка
             List.Delete(i);
             i:=i-1;
          end
          else begin
             // MD-файл существует
             Yes:=1;
          end;
        end
        else begin
          // Без проверки существования файла 1Cv7.md
          Yes:=1;
        end;
        if Yes>0 then begin
          // Читаем имя базы данных
```

```
            sNameDB:=Trim(Reg.ReadString(List[i]));
            List[i]:=sNameDB+'='+List[i];
          end;
          Application.ProcessMessages;
         end;
         //--------------------------------
        end;
      end;
      Result:=List.Count;
     finally
      // Освобождаем ресурсы
      Reg.Free;
     end;
   end;
 end;
end;
```

Поясним листинг 12.1.

Здесь после открытия соответствующего раздела системного реестра:

```
if Reg.OpenKey(KeyName,False) then begin
```

сначала считывается список каталогов (папок), где размещаются зарегистрированные базы данных 1С:

```
Reg.GetValueNames(List);
```

Затем, в зависимости от значения параметра `YesDetect_1Cv7md`, либо сразу формируется список баз данных:

```
sNameDB:=Trim(Reg.ReadString(List[i]));
List[i]:=sNameDB+'='+List[i];
```

либо сначала проверяется наличие файла конфигурации 1Cv7.md в каталоге базы данных. И если такого файла нет, предполагается, что тестируемая база данных некорректна и удаляется из списка:

```
// Проверяем, существует ли MD-файл
if not FileExists(fnMD) then begin
  // MD-файл НЕ существует, удаляем базу данных из списка
  List.Delete(i);
  i:=i-1;
end
else begin
```

```
  // MD-файл существует
  Yes:=1;
end;
```

**ПРИМЕЧАНИЕ**

Исходный текст примера, реализующего данный алгоритм, приведен на сопроводительном компакт-диске книги в папке \Part_3\Chapter_12\LoadListDB_1C.PAS\ (файл LoadListDB_1C.DPR).

На рис. 12.3 показан результат считывания из реестра Windows списка баз данных системы 1С:Предприятие.

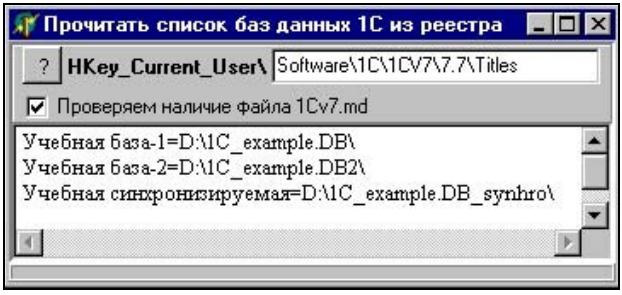



**Рис. 12.3.** Список конфигураций (баз данных) 1С, прочитанных из системного реестра Windows

# Глава 13

# Список активных пользователей базы данных 1С:Предприятие

В процессе эксплуатации системы 1С:Предприятие может возникнуть необходимость контроля пользователей, которые в текущий момент времени работают с определенной базой данных 1С (активные пользователи). Это может быть важно в тех случаях, например, когда требуются какие-либо регламентные процедуры по обслуживанию базы данных, возможные только при отсутствии активных пользователей. Можно, конечно, воспользоваться Монитором 1С (рис. 13.1).

| | Имя пользователя | Начало работы | Компьютер |
|---|---|---|---|
| | Admin | 11.11.2006,14:08:17 | COMP |
| | Пользователь_1 | 11.11.2006,14:08:07 | COMP |
| | Пользователь_2 | 11.11.2006,14:07:56 | COMP |
| | Пользователь_3 | 11.11.2006,14:07:49 | COMP |

**Рис. 13.1.** Список активных пользователей базы данных (Монитор 1С:Предприятие)

Но иной раз (для автоматизации этого процесса) предпочтительнее разработать собственные программные средства (например, для рассылки соответствующих оповещений о выходе из 1С всем активным пользователям базы данных). Естественно, лучше всего это делать вне контекста программы 1С:Предприятие — хотя бы затем, чтобы лишний раз не занимать ресурсы самой 1С для такой чисто системной задачи.

В каталоге каждой базы данных 1С есть подкаталог SYSLOG, где (в случае, если в данной конфигурации используется система разграничения прав доступа к информационной базе) присутствует файл links.tmp. Именно в этом файле и хранится информация об активных пользователях конкретной базы данных. Этот файл появляется в момент загрузки конфигурации 1С после авторизации пользователя и может быть автоматически удален программой 1С:Предприятие после ее закрытия, если больше нет активных пользователей.

Если загрузить этот файл в любой визуальный Delphi-компонент (имеющий свойство `TStrings`), то видно (рис. 13.2), что информация в файле хранится в SDF-формате *(см. главу 10)*. Каждому пользователю отведено четко определенное количество строк файла (размер области файла, выделяемой для каждого пользователя, — 1024 байта).

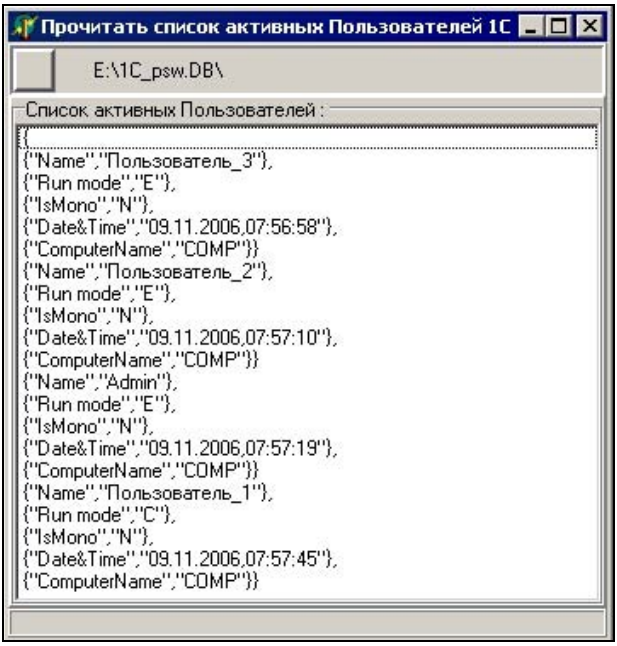



**Рис. 13.2.** Содержание файла links.TMP

Как видим, выделить из этого файла нужную информацию и конвертировать ее в любой другой формат (для целей визуализации) не составит труда. Но есть один существенный нюанс…

Если провести простой эксперимент (любой желающий может сделать это на локальной базе данных 1С), то видно, что с каждым новым подключением пользователя к конкретной базе данных 1С файл links.TMP растет (по 1024 байта на одно подключение, даже если один и тот же пользователь подключился несколько раз), но по мере отключения пользователей размер файла не меняется и пользователи из списка (см. рис. 13.2) не исчезают. То есть для получения списка активных пользователей одной этой информации недостаточно.

Программа 1С:Предприятие (и Конфигуратор 1С) "помечает" каждого активного пользователя следующим образом. В файле links.TMP блокируется байт

со смещением `1E8481h + НомерПользователяВСписке`. При этом нумерация пользователей начинается с нуля. Поэтому чтобы определить, является ли пользователь в списке (см. рис. 13.2) активным, достаточно сделать попытку блокировки соответствующего байта в файле links.TMP. Если удачно — этот Пользователь *не* активен.

В листинге 13.1 приведен текст функции, которая возвращает список активных пользователей, применяя этот способ.

**Листинг 13.1. Функция чтения списка активных пользователей базы данных 1С**

```
function Get_List_ActiveUsers_from_1C(DirTMP,
                                fn_Links_TMP : string;
                                ListUsers    : TStrings) : integer;
// Получение списка активных Пользователей 1С:
// DirTMP       - имя каталога для временных файлов
//                (должен существовать),
// fn_Links_TMP - полное имя файла Links.TMP
//                (в тестируемой конфигурации 1С),
// ListUsers    - список активных пользователей
//                (каждая строка списка в SDF-формате).
Var
  fn_Links_TMP_local : string;
  fHandle            : THandle;
  i                  : integer;
  Sx                 : string;
  YesNext            : byte;
  List               : TStrings;
begin
 Result:=0;
 YesNext := 0;
 if ListUsers<>NIL then begin
   ListUsers.Clear;

   // Создаем временный список пользователей
   List := TStringList.Create;
   TRY

    DirTMP:=Trim(DirTMP);
    if length(DirTMP)>0 then begin
```

```
if DirTMP[length(DirTMP)]<>'\' then begin
  DirTMP:=DirTMP+'\';
end;
// Проверяем наличие папки <DirTMP>
if DirectoryExists(DirTMP) then begin
  // Проверяем наличие файла <Links.tmp>,
  // в каталоге тестируемой конфигурации
  if FileExists(fn_Links_TMP) then begin
    // Вычисляем новое (полное) имя файла Links.tmp,
    // куда скопируем тестируемый файл
    fn_Links_TMP_local:=DirTMP+ExtractFileName(fn_Links_TMP);
    // Копируем тестируемый файл Links.tmp
    // в папку DirTMP
    if Windows.CopyFile(PCHAR(fn_Links_TMP),
                        PCHAR(fn_Links_TMP_local),
                        FALSE) then begin
      // Загружаем файл <Links.tmp> из
      // папки DirTMP в List
      List.LoadFromFile(fn_Links_TMP_local);
      if List.Count>0 then begin
        // Формируем список всех пользователей
        // из файла links.tmp
        //----------------------------
        // Удаляем все "лишние" строки,
        // кроме имени пользователя
        i:=-1;
        while i<(List.Count-1) do
         begin
         i:=i+1;
         if POS('{"Name"',List[i])<>1 then begin
           List.Delete(i);
           i:=i-1;
          end;
        end;
        //----------------------------
        if List.Count>0 then begin
          // Продолжаем, если список НЕ пуст
          YesNext := 1;
```

```
            end;
          end;
        end;
      end;
    end;
  end;
  if YesNext>0 then begin
    // Открываем файл Links.tmp
    fHandle:=Windows.CreateFile(PCHAR(fn_Links_TMP),
                        GENERIC_READ or GENERIC_WRITE,
                        FILE_SHARE_READ OR FILE_SHARE_WRITE,
                        0,
                        OPEN_EXISTING,
                        FILE_ATTRIBUTE_NORMAL,
                        0);
    TRY
     //------------------------
     // Проверяем блокировку пользователей
     i:=-1;
     while i<(List.Count-1) do
      begin
      i:=i+1;
      // Пытаемся блокировать область в файле Links.tmp,
      // соответствующую данному пользователю
      if not (LockFile(fHandle,
                       $1E8481+i, 0, 1, 0)) then begin
        // Неудачная блокировка - значит, этот пользователь
        // активен, добавляем его в наш список
        ListUsers.Add(List[i]);
      end
      else begin
        // Удачная блокировка - значит, этот пользователь
        // НЕ активен. Освобождаем блокировку
        UnLockFile(fHandle, $1E8481+i, 0, 1, 0);
      end;
      Application.ProcessMessages;
     end;
     //------------------------
```

```
        // Вычисляем возвращаемое значение функции
        Result:=ListUsers.Count;
       FINALLY
        // Освобождаем ресурсы
        Windows.CloseHandle(fHandle);
       END;
     end;
    FINALLY
     // Освобождаем ресурсы
     if List<>NIL then begin
        List.Clear;
        List.Free;
        List:=NIL;
     end;
    END;
  end;
end;
```

Поясним листинг 13.1. Здесь сначала формируется временный список всех пользователей (`List`), указанных в файле links.tmp. Затем для каждого пользователя делается попытка блокировки соответствующего байта. Если блокировка *не* была успешной, считается, что этот пользователь подключен к базе данных, и информация о нем копируется в список активных пользователей (`ListUsers`).

На рис. 13.3) представлен результат, возвращаемый функцией `Get_List_ActiveUsers_from_1C`.

Естественно, в таком виде список не очень-то легко читается.

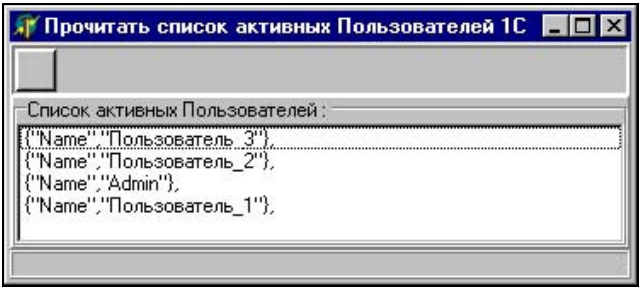



**Рис. 13.3.** Список активных пользователей базы данных 1С

В листинге 13.2 представлена функция, конвертирующая список (см. рис. 13.3) в такой же формат, как на рис. 13.4.

**Листинг 13.2. Функция, извлекающая из списка активных пользователей их имена**

```
function Exctract_UserName_from_ListActiveUsers(
                                          ListUsers:TStrings):integer;
// Извлечение имен пользователей из списка активных пользователей 1С,
// полученного с помощью функции Get_List_ActiveUsers_from_1C,
Var
  List : TStrings;
  i : integer;
begin
 Result:=0;
 if ListUsers<>NIL then begin
   if ListUsers.Count>0 then begin
     List := TStringList.Create;
     TRY
      // Сканируем список пользователей
      for i:=0 to (ListUsers.Count-1) do
       begin
        // Разбиваем строку списка на части
        sp_Convert_SDF_to_List(ListUsers[i],List,'{} ');
        // Выбираем вторую часть - имя пользователя
        ListUsers[i]:=GetValueFromList(2,List);
      end;
      Result:=ListUsers.Count;
     FINALLY
      // Освобождаем ресурсы
      if List<>NIL then begin
        List.Clear;
        List.Free;
        List:=NIL;
      end;
     END;
   end;
 end;
end;
```

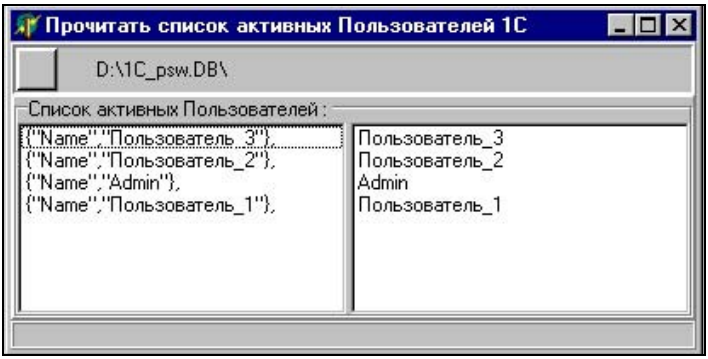



**Рис. 13.4.** Список активных пользователей базы данных 1С после конвертации

Текст функции `Exctract_UserName_from_ListActiveUsers` достаточно прост и вряд ли требует пояснений. Функции `sp_Convert_SDF_to_List` и `GetValueFromList` были подробно рассмотрены в *главе 10* и здесь их тексты не приводятся.

***Примечание***

Текст этого примера приведен на сопроводительном компакт-диске книги в папке \Part_3\Chapter_13\LoadListActiveUsers_1C.PAS\LoadListActiveUsers_1C.DPR.

В папке \Part_3\Chapter_13\GetUsers1C.PAS\ сопроводительного компакт-диска книги приведены исходные тексты программы GetUsers1C.EXE, назначение которой:

- считывание списка зарегистрированных баз данных 1С из системного реестра (с возможностью считывания списка баз данных из текстового файла);
- периодическое (по таймеру) считывание списка активных пользователей для заданной конфигурации (базы данных) 1С;
- отсылка оповещения по сети на компьютер активного пользователя;
- отсылка оповещения по сети на заданный компьютер, если список активных пользователей уже пуст.

В программе GetUsers1C.EXE применены все способы обработки информации, рассмотренные нами ранее. На рис. 13.5 приведен результат визуализации списка активных пользователей.

| N | User | Date | Time | Computer | IsMono | Run Mode |
|---|---|---|---|---|---|---|
| 1 | Пользователь_3 | 06.11.2006 | 14:56:25 | COMP | N | E |
| 2 | Пользователь_2 | 06.11.2006 | 14:56:44 | COMP | N | E |
| 3 | Admin | 06.11.2006 | 14:56:51 | COMP | N | E |
| 4 | Пользователь_1 | 06.11.2006 | 14:57:03 | COMP | N | E |

**Рис. 13.5.** Список активных пользователей базы данных 1С (программа GetUsers1C.EXE)

# Глава 14

# Отслеживание регламента системного обслуживания баз данных 1С:Предприятие

Как было отмечено в *части I*, в процессе производственной эксплуатации системы 1С:Предприятие может потребоваться периодическое и оперативное системное обслуживание баз данных 1С (например, изменение информационной структуры конфигурации). А это возможно лишь в том случае, если в контексте данной конфигурации нет активных пользователей. Следовательно, возникает необходимость в создании некоего регламента, который должен обеспечить освобождение баз данных 1С (т. е. выход пользователей из 1С) в строго заданные отрезки времени суток (часто — на время обеденного перерыва).

Создать-то регламент можно, и даже утвердить его в самых высоких инстанциях (на предприятии) с указанием мыслимых и немыслимых карающих мер по отношению к "тем, кто" — тоже несложная задача. Но, как показывает опыт, отслеживать этот регламент при большом количестве пользователей очень и очень проблематично.

Пользователь — он и в Африке пользователь. У пользователей своих проблем (в предметной области) по горло. И, по сути являясь существами в целом сознательными и доброжелательными, часто хотя бы один-другой пользователь (срочно убегая в столовую на обед), но забудет закрыть программу 1С перед уходом, со всеми вытекающими… Думаю, вряд ли кто-то сможет предложить одну "универсальную таблетку" для решения этой проблемы. Скорее всего здесь нужен ряд эффективных мер, специфичных для каждого предприятия. Но в любом случае необходим соответствующий программный инструментарий, помогающий отслеживать утвержденный регламент. В предыдущей главе мы рассмотрели способ выявления списка активных пользователей в заданной конфигурации, что может быть полезно при решении этой задачи.

Посмотрим, что можно еще предпринять для этих целей. Ранее *(в части I)* мы рассматривали способ визуализации окна программы 1С:Предприятие (как процесса в среде Windows). Мы создали функции, определявшие наличие программы 1С:Предприятие в памяти компьютера. В принципе, нетрудно написать программу, которая будет автоматически запускаться при запуске Windows и периодически (например, раз в минуту) проверять наличие программы 1С:Предприятие. Если в такой программе предусмотреть возможность навязчивого напоминания пользователю (минут за 5—10 до начала регламентного перерыва) о том, что он должен выйти из 1С, то это снизит процент "забывчивых".

***Примечание***

На сопроводительном компакт-диске книги в папке \Part_3\Chapter_14\wSpy1C.PAS\ приведены исходные тексты программы wSpy1C.EXE, ориентированной как раз на такую ситуацию.

Здесь мы не будем подробно рассматривать тексты программы wSpy1C.EXE (кто желает — посмотрит самостоятельно). Отметим только, что для сканирования по таймеру списка окон приложений, открытых в среде Windows, используются стандартные функции API Windows (листинг 4.98) — `GetTopWindow` и `GetNextWindow`. Текст заголовка окна считывается (чтобы идентифицировать окно 1С:Предприятие) с помощью функции `GetWindowText`. В остальном — что подскажет программисту его фантазия. Программа wSpy1C.exe, например, очень навязчиво выдает соответствующее предупреждение на экран монитора пользователя (рис. 14.1) до тех пор, пока он не закроет программу 1С:Предприятие.

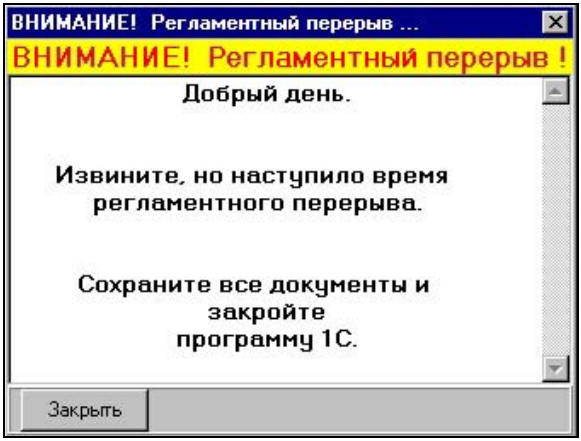



**Рис. 14.1.** Предупреждение о наступлении времени регламентного перерыва (программа wSpy1C.exd)

Возможны ситуации, когда (при запуске программы 1С) на экран выводится диалоговое окно, и программа ожидает идентификации пользователя. В этот момент пользователь уже "вошел" в систему, но в Мониторе (1С) его не видно (справедливости ради следует отметить, что и способ контроля активных пользователей, рассмотренный в *главе 13*, также не эффективен). То есть пользователь отправился по своим делам, диалоговое окно авторизации (идентификации) "висит" на экране, Монитор 1С ничего не видит, но база данных 1С занята. Сохранить изменения, сделанные в конфигурации, невозможно…

Вычислить такую ситуацию бывает проблематично. Программа wSpy1C.EXE "борется" с этой ситуацией следующим образом:

1. Наряду с процессом поиска окна программы 1С:Предприятие (в заголовке окна присутствует строка символов `"1С:Предприятие"`) производится поиск окна идентификации пользователя с заголовком (листинг 14.1):

   `"1Cv7 – Загрузка конфигурационной информации..."`
2. В случае, если такое окно найдено, программа wSpy1C.EXE пытается его закрыть, эмулируя соответствующие действия пользователя (листинг 14.2).

**Листинг 14.1. Функция поиска открытого диалогового окна авторизации**

```
function GetHandleForDialogPassword1C : hWND;
// Функция проверяет :
// "висит" ли в памяти диалоговое окно ввода пароля для входа в 1С.
// Если ДА, то возвращает дескриптор окна
Var
 Ref1,Ref2 : hWND;
begin
 Result:=0;
 Ref1:= FindWindow(nil,'1Cv7 - Загрузка конфигурационной информации...');
 if Ref1>=32 then begin
    Ref2:=FindWindow(nil,'Авторизация  доступа');
    if Ref2>=32 then begin
       Result:=Ref2;
    end;
 end;
end;
```

**Листинг 14.2. Функция, эмулирующая действия пользователя по закрытию диалогового окна авторизации**

```
function CloseDialogPassword1C : boolean;
// Закрыть диалог ввода пароля для входа в 1С
Var
 Ref : hWND;
begin
 Result:=FALSE;
 Ref:=GetHandleForDialogPassword1C;
 if Ref>=32 then begin
    SendDlgItemMessage(Ref,2,WM_KEYDOWN,32,0);
    SendDlgItemMessage(Ref,2,WM_KEYUP,32,0);
    Result:=TRUE;
 end;
end;
```

Тексты обеих функций "прозрачны" и не требуют пояснений.

# Глава 15

# Программа 1С:Предприятие в качестве DDE-сервера

DDE (Dynamic Data Exchange — динамический обмен данными) — это протокол, поддерживающий передачу данных между приложениями, функционирующими в среде Windows. При динамическом обмене данными одна программа (которая запрашивает данные) становится *DDE-клиентом*, а другая (которая поставляет данные) — *DDE-сервером*. DDE-протокол основан на использовании девяти сообщений (табл. 15.1).

***Таблица 15.1.*** *Сообщения DDE-протокола обмена данными*

| Сообщение | Назначение |
|---|---|
| `wm_DDE_Initiate` | Начать сеанс обмена данными |
| `wm_DDE_Terminate` | Завершить сеанс обмена данными |
| `wm_DDE_Execute` | Выполнить команду на сервере |
| `wm_DDE_Ack` | Получить статус операции |
| `wm_DDE_Poke` | Послать данные серверу |
| `wm_DDE_Request` | Запросить данные с сервера |
| `wm_DDE_Advise` | Установить связь с сервером |
| `wm_DDE_UnAdvise` | Завершить связь с сервером |
| `wm_DDE_Data` | Передать данные клиенту |

Обмен данными (по DDE-протоколу) состоит из посылки ряда сообщений, общая форма которых имеет вид:

```
PostMessage(hWndR, wm_DDE_XXX, hWndS, Data);
```

Параметры: `HWndR` — дескриптор (ссылка) окна программы-клиента; `wm_DDE_XXX` — сообщение DDE; `hWndS` — дескриптор (ссылка) окна программы-сервера; `Data` — данные (атомы, ссылки на глобальные данные и т. д.).

DDE-сеанс начинается с того, что программа-клиент (инициатор сеанса) определяет название приложения (сервера) и темы "разговора". На основе DDE-протокола построен протокол OLE.

Программа 1С:Предприятие может выполнять роль DDE-сервера, предоставляя свой сервис в получении данных (в частности, функции глобального модуля, объявленные с ключевым словом `Экспорт`). Программа MS Excel может выполнять роль как DDE-сервера, так и DDE-клиента. Поэтому попробуем использовать Excel (в качестве DDE-клиента) для иллюстрации возможностей 1С:Предприятие (в качестве DDE-сервера).

Запустим программу 1С:Предприятие и откроем "нашу" конфигурацию 1С_example.DB. Затем запустим Excel.

***Примечание***

Роль DDE-клиента может выполнять любая ячейка Excel.

В документации к программе 1С:Предприятие приводится пример считывания рабочей даты в системе:

```
='1CV7'|DDE!'РабочаяДата()'
```

Попробуем таким же образом прочитать текущую дату:

```
='1CV7'|DDE!'ТекущаяДата()'
```

Результат представлен на рис. 15.1.

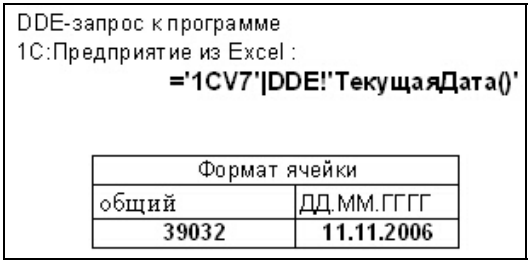



**Рис. 15.1.** Результат считывания системной даты с использованием DDE-протокола

Теперь попробуем вызвать на выполнение функции, объявленные в глобальном модуле "нашей" конфигурации с ключевым словом `Экспорт`.

Запрос:

```
='1CV7'|DDE!'гл9_А_плюс_Б(5,7)'
```

Ответ: 12.

Запрос:

```
='1CV7'|DDE!'гл9_ЗначениеВСтрокуВнутр_для_Перечисления("Пол",1)'
```

Ответ: `{"E","0","0","17","0","0","  19  "}`.

Запрос:

```
='1CV7'|DDE!'гл9_ПолучитьЗначРеквизитаСправочника("ФизЛица","ГородПрожива
ния",1,"",0)'
```

Ответ: `Усинск`.

> ***Примечание***
>
> На сопроводительном компакт-диске книги в папке \Part_3\Chapter_15\DDE\ есть файл 1C_DDE.XLS (Excel), где приведены все эти примеры.

В свое время я был заинтересован возможностью использовать механизм DDE-сообщений для того, чтобы можно было в реальном времени отслеживать начало каких-то процессов в системе 1С:Предприятие (например, проведение документов) и их окончание. И, таким образом, построить некую "событийную" систему извещений пользователей о начале и окончании определенных процессов в системе. Но, к сожалению, не нашел в документации к 1С никакой информации о механизме уведомления (со стороны 1С как DDE-сервера) DDE-клиента об изменении состояния системы 1С:Предприятие.

Естественно, рассмотренные в этой части примеры обмена информацией с системой 1С:Предприятие, реализованные как приложения в среде Windows, — лишь малая часть того, что можно использовать для решения прикладных задач системного сопровождения программы 1С:Предприятие. Но мы на этом остановимся.

> ***Примечание***
>
> Напомню, что представленные примеры исходных текстов — это упрощенная иллюстрация к рассматриваемым вопросам. Назначение этих примеров — дать представление о возможном подходе при решении задач информационного обмена с системой 1С:Предприятие. При работе с реальными базами 1С может потребоваться их доработка.

# Приложение

# Описание компакт-диска

Структура папок на сопроводительном компакт-диске книги соответствует ссылкам на примеры, приведенные в тексте книги. Каждый пример находится в отдельной папке. Каждая папка с примером расположена в папке, соответствующей номеру части, главы, раздела и подраздела. Если пример имеет несколько версий, то в папке примера находятся подпапки, имена которых соответствуют номеру версии. Каждая папка примера (или версии примера) содержит файлы, список которых определяется тематикой и контекстом изложения в книге.

Например, в *разд. 4.7.1 части I* приведен пример редактирования реквизитов справочника с использованием механизма OLE Automation. Файлы версии 1 этого примера расположены, соответственно, в папке \Part_1\Chapter_04\Topic_4_7\Item_4_7_1\sp_EditAttrInRef.PAS\01.

## Структура содержимого компакт-диска

Сопроводительный компакт-диск книги содержит папки:

- 1C_example.DB — учебная конфигурация 1С; основная тренировочная;
- 1C_example.DB_synhro — учебная конфигурация 1С; иллюстрация процесса синхронизации конфигураций;
- 1C_example.DB2 — учебная конфигурация 1С; иллюстрации использования возможностей внепроцессного сервера автоматизации (oleSrv1C.EXE) для проверки дат на корректность;
- sp_Lib — исходные тексты библиотечных функций, используемых в программных примерах, приведенных на сопроводительном компакт-диске;

- Part_1 — примеры по теме *части I* "1С и механизм OLE Automation":
  - Chapter_03 — примеры по теме *главы 3* "Подключение к 1С";
  - Chapter_04 — примеры по теме *главы 4* "Доступ к информации 1С из внешних приложений";
  - Chapter_05 — примеры по теме *главы 5* "Расширение функционала 1С";
- Part_2 — примеры по теме *части II* "Доступ к базе данных 1С без программы 1С:Предприятие":
  - Chapter_06 — примеры по теме *главы 6* "Файлы формата DBF";
  - Chapter_07 — примеры по теме *главы 7* "Обработка словаря базы данных 1С (файл 1Cv7.DD)";
  - Chapter_11 — пример по теме *главы 11* "Остатки ТМЦ на складах";
- Part_3 — пример по теме *части III* "Программа 1С:Предприятие в среде Windows":
  - Chapter_12 — примеры по теме *главы 12* "Список зарегистрированных баз данных 1С:Предприятие";
  - Chapter_13 — примеры по теме *главы 13* "Список активных пользователей базы данных 1С:Предприятие";
  - Chapter_14 — примеры по теме *главы 14* "Отслеживание регламента системного обслуживания баз данных 1С:Предприятие";
  - Chapter_15 — примеры по теме *главы 15* "Программа 1С:Предприятие в качестве DDE-сервера".

# Предметный указатель


**1**

1Cv7.MD, файл 227

**A**

ActiveForm, значок 429
ActiveX Library 362
ActiveX Library, значок 363
AddIn_Example02_01, внешняя компонента 420
AddIn_Example02_02, внешняя компонента 429
AddIn_Files, внешняя компонента 414, 415, 416
ANSI (American National Standards Institute) 452
Automation Object 363, 434

**B**

BasicScript 446
BDE-5, пакет 448, 457

**C**

C++Script 446
CallAsProc, метод 389
COM (Component Object Model 360
COM-объект 360
COM-сервер 361

**D**

dBASE 453
dBASE III+ 454
DBF, формат 13, 453
DBF-файл 262, 479
  структура 462
DBT-файл 456
DDE (Dynamic Data Exchange) 570
DDE-клиент 570
DDE-сервер 570
Detail table 51
DLL 371

**E**

EvalExpr, метод 234, 243
Excel 262
ExecuteBatch, метод 234, 262

**F**

FALSE 404, 415
FastScript, пакет 446
FindByCode, метод 49

**G**

Generate C object files, опция 383
GetFieldName_from_1Cv7DD, функция 470

GetFieldValue_from_TableDB, функция 511
GetSbCntName, функция 510
GetUsers1C.EXE, программа 565
GetWindowText, функция 567
GUID 360, 371

**H**

HResult, тип 361

**I**

IAsyncEvent, интерфейс 389
IDispatch, интерфейс 426
IDL (Interface Definition Language) 362
IExtWndsSupport, интерфейс 419
INI-файл 415, 479
In-process 361
IPropertyLink, интерфейс 380
IStatusLine, интерфейс 406

**J**

JScript 446

**L**

Links.TMP, файл 558
Lock-bytes 216
LRPC (Local Remote Procedure Calls) 433

**M**

Main MetaData Stream 484, 485—490
Master table 51

**O**

OLE Automation 178
oleVariant, тип 316
OLE-объект внешней компоненты 375
Out of process 361

**P**

PascalScript 446
Property Page, значок 367
public, секция 244

**R**

RA, префикс 163
Remote 361
RG, префикс 163
Root storage 219

**S**

Safecall, вызов 361
Sample 368
Sample2 419
SDF, формат 559
SEQUEL 452
SP 508
sp_GetFromINI_StringParam, функция 481
sp_GetWordFromString, функция 149
SQL (Structured Query Language) 452
SQL-3 452
SQL-подзапрос 543
Storage 216
Stream 216
String, тип 152

**T**

TAddInObject, объект 393, 399
TMethods, перечисление 399
TRUE 404, 415

**V**

ValueToStringInternal, функция 207
Variant, тип 316

**W, Y**

wSpy1C.EXE, программа 567
YesAttrInDoc_1C, функция 137

## А

Авторизация 568
Агрегатный тип 152, 522
Активные пользователи 558
  список 565

## Б

Библиотека типов 422
Блок информационных
  записей DBF-файла 453
Блок описания таблицы 462
Блокированные данные 216

## В

ВвестиПеречисление, функция 241
Версия внешней компоненты 372
Видимость окна 437
Визуализация справочника 283
Включить, метод 383
Владелец 52
Вложенный SQL-подзапрос 543
Внепроцессный сервер 361, 433,
  444, 445
Внутренний системный
  идентификатор 207
Внутрипроцессный сервер 361, 444
Время документа 150
Время считывания 178
Выбор:
  информации из справочника 39
  нужного документа 140
Выборка элементов справочника 56
Выключить, метод 383

## Г

гл9_СписокЗначений_
  ДобавитьЗначение, функция 239
гл9БуферДляOLE, переменная 234
Главная таблица 51
Глобальный контекст 203
Глобальный модуль 182, 207, 384
Группа 352
Группировка 81

## Д

Дата:
  формат 437
Движения 178
Движения документа 162
Дерево 498
Дескриптор:
  окна 245, 427
  поля таблицы 455
Диалоговые окна 419
Динамический обмен данными 570

## Ж

Журналы документов 144

## З

Заголовок:
  DBF-файла 453
  окна 426
  таблицы, структура 454
ЗагрузитьВнешнююКомпоненту,
  метод 384
Записная книжка 428
Запрос 258
Зарегистрированные базы
  данных 1С, список 565
Значение:
  объекта 8
  перечисления 40
  поля 511
  периодического реквизита 324
  реквизита справочника 40
  реквизита, чтение 155, 441
  свойства 379
ЗначениеВСтроку,
  функция 316, 319
ЗначениеВСтрокуВнутр,
  функция 207
ЗначениеИзСтроки, функция 316

## И

Идентификатор 9
  документа 148
  значения 33
  перечисления 33
  справочника 9
  элемента управления 386
Идентификаторы:
  измерений регистра 167
  регистров
    список 164
Иерархическая структура 193
Иерархия 67
  проверка 84
Изменение идентификатора реквизита 138
Измерения регистра 157, 166
Имена свойств и методов внешней компоненты 374
Имя диска 414
Имя:
  метода 378
  свойства 377
  таблицы 463
  файла 414
Инвертирование строки 210
Индексы 56
Интернет-ресурсы 227
Интерфейс 360
  IDispatch 361
  IExtWndsSupport 419
  IpropertyLink 380
  IStorage 219
История изменений значения периодического реквизита 96

## К

Каталог (папка) 383
Кириллица 6
Клавиши для визуализации окна 186
Клиент автоматизации 3
Код объекта 8
Код, свойство 11
Кодировка 6
Количество измерений регистра 166
Количество:
  методов внешней компоненты 375
  регистров 163
  свойств внешней компоненты 374
  строк 150
Компоненты в форме проекта 188
Контроллер автоматизации 3, 368, 436
Конфигуратор 4
Корневой каталог 219

## Л

Лист дерева 501
Логическое значение 404

## М

Маска 398, 415
Маска файлов 414
Материалы 526
Материально-производственные запасы (МПЗ) 526
Метаданные 63, 84
Метод:
  CallAsProc 389
  EvalExpr 234, 243
  ExecuteBatch 234, 262
  FindByCode 49
  GetFieldName_from_1Cv7DD 470
  GetFieldValue_from_TableDB 511
  GetSbCntName 510
  GetWindowText 567
  sp_GetFromINI_StringParam 481
  Включить 383
  Выключить 383
  ЗагрузитьВнешнююКомпоненту 384
  ЗначениеВСтроку 316
  ЗначениеИзСтроки 316
  НазначитьВид 306
  НазначитьТип 294
  НайтиПоКоду 49
  ПодключитьВнешнюю Компоненту 384
  ПолучитьАтрибут 152
  СоздатьОбъект 236
  УстановитьАтрибут 313

Методы внешней компоненты 369
Модель:
разделенных потоков 365
свободных потоков 365
смешанных потоков 365
составных объектов 360
МПЗ 526

## Н

НазначитьВид, метод 306, 307
НазначитьТип, метод 294
Наименование, свойство 11
НайтиПоКоду, метод 49
Неопределенный тип и вид 521
Непериодические реквизиты 41, 327
Номенклатурный справочник 526
Номер:
документа 148
метода 377
свойства 376

## О

Оборотный регистр 157
ОбработкаПроведения, процедура 180
Общие реквизиты документов 151
Общие свойства документа 150
Объект метаданных 203
Однопоточная модель 365
Окно программы 1С:Предприятие 186
Описатель:
индексов таблицы 463
полей таблицы 455, 463
таблицы 462
Определение типа значения реквизита 289
Открытие формы 264

## П

Пакет BDE-5 457
Палитры компонентов Delphi 459
Переменная объектного типа 360
Перечисления 32
идентификатор 33
Периодические реквизиты 86, 325, 520
Подключение внешней компоненты 384
ПодключитьВнешнююКомпоненту, метод 384
Подчиненная таблица 51
Подчиненный справочник 51
Позднее связывание 361
Позиция документа 150
ПолучитьАтрибут, метод 152
Поток 216
Предметы труда 526
Представление документа 149
Преобразование значения объекта агр. типа в строковое 207
Преобразование строки 233
Признак актуальности записи 454
Признак проведения документа 150
Приход 174
Проверка:
присутствия идентификатора 173
существования измерения 172
существования реквизита 63, 132, 133, 134, 135, 136
существования справочника 63
Программа:
GetUsers1C.EXE 565
wSpy1C.EXE 567

## Р

Разделитель 522
Размеры и положение окна 427
Разрезание файла 415
Раннее связывание 361
Расход 174
Расширение функциональности 433
Регистр остатков 157, 179
Регистрация сервера в ОС 434
Регистры:
доступ из 1С 157
количество 163
Регламент 566
Редактирование реквизита 279
Реестр 554

Реквизит:
изменение идентификатора 138
неопределенный 301
Реквизиты:
шапки документа 152
документа 151, 333
3 категории 124
регистра 157
табличной части документа 152
Реляционная СУБД 453
Ресурсы регистра 157
Родитель 81, 351

## С

Свойства:
внешней компоненты 373
документа 110, 124, 150
Свойства и методы внешней компоненты 374
Связь между таблицами, схема 533
Секция public 244
Сервер:
OLE Automation 3
автоматизации 3
внепроцессный 361
внутрипроцессный 361
удаленный 3, 361
Сервисы 360
Сиквел 452
Синхронизация справочников 349
Системный идентификатор 207
значения перечисления 212
объекта 211
Склейка файла 415
Скрипт 445
Словарь данных 46, 462
Создание форм 419
СоздатьОбъект, метод 236
Сообщения 389, 390, 398, 449
Сортировка 57
Сохранение:
значений элементов массива 243
информационной структуры 1С 479
Список:
баз данных 554
дескрипторов таблиц 466
идентификаторов измерений регистра 167
идентификаторов регистров 164
окон в Windows 245
таблиц 466
Способы чтения значения реквизита 441
Справочник 7
Сравнение объектов в 1С 38
Ссылка на документ 150, 174
Стандарт COM 360
Стиль окна 426
Страница свойств 381
Строка неограниченной длины 42
Строка состояния 398
Структура:
информационной записи 455
регистра 158
таблицы 467
СУБД 453
Схема связи между таблицами 533
Сценарий 445
Считывание идентификаторов:
значений перечисления 201
значения реквизита 285
остатков 179
списка документов 196
Считывание свойств:
документов 195
измерения регистра 198
константы 200
регистра 198
реквизита регистра 200
реквизитов "шапки" 197
реквизитов справочника 194
ресурса регистра 199
справочников 193

## Т

Таблица в формате DBF 456
Таблицы, схема связи 533
Табличная часть документа 333
Таймер 393, 565

Текстовый файл 254, 262
Терминальный байт 453
Технология создания внешних компонент 359
Тип 485—488
  значения 152
  поля 455
Типы данных 1С 29
Типы окон 419
Товарно-материальные ценности (ТМЦ) 526

## У

Удаленный сервер 3, 361
Узел дерева 501
УстановитьАтрибут, метод 313

## Ф

Фабрика класса 361
Файл:
  1SBLOB.DBF 47
  links.TMP 558
  в формате DBF 456
Флаг доступности 381, 382
Форма редактирования 42
Формат:
  DBF 13
  кодировки 6
Формы 419
  создание 419

## Х, Ч

Хранилище 216
Чтение из справочника 238

## Ш

Шаблон 368
Шапка документа 124, 333

## Э

Экземпляр 360
Экспорт 262
Элемент справочника 98

## Я

Язык IDL 362
Язык структурированных запросов 452