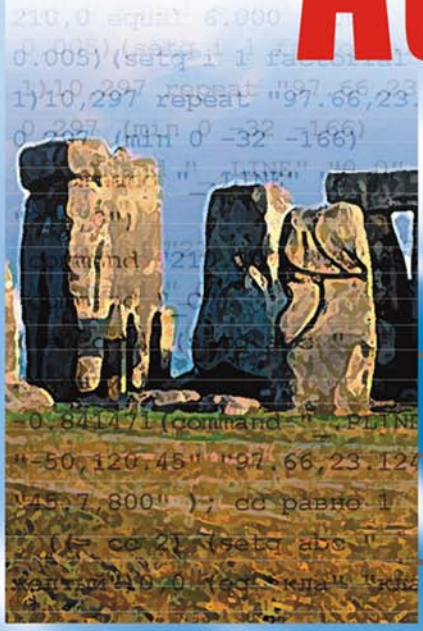
Николай Полещук,
Петр Лоскутов

bhv®

AutoLISP и Visual LISP в среде AutoCAD



Наиболее полное руководство

+CD

В ПОДЛИННИКЕ®

Николай Полещук
Петр Лоскутов

AutoLISP и Visual LISP в среде AutoCAD

Санкт-Петербург
«БХВ-Петербург»
2006

УДК 681.3.068
ББК 32.973.26-018.1
П49

Полещук Н. Н., Лоскутов П. В.

П49 AutoLISP и Visual LISP в среде AutoCAD. — СПб.: БХВ-Петербург, 2006. — 960 с.: ил.

ISBN 5-94157-738-9

Книга является руководством разработчика LSP-, FAS- и VLX-приложений, функционирующих в среде графического процессора AutoCAD версий 2005 и 2006. Содержит полную информацию по языкам AutoLISP и Visual LISP. Освещает вопросы COM-связи с внешними приложениями, программной адаптации интерфейса, создания справочных систем. Впервые на русском языке полностью описаны и пояснены на примерах методы и свойства объектной модели AutoCAD, используемой в технологии ActiveX. Исходные тексты рассмотренных программ размещены на сопроводительном компакт-диске.

Для программистов и разработчиков приложений в среде AutoCAD

УДК 681.3.068
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Наталья Таркова*
Зав. редакцией	*Григорий Добин*
Редактор	*Вильга Савельева*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн серии	*Игоря Цырульникова*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 22.03.06.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 77,4.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

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

ISBN 5-94157-738-9

© Полещук Н. Н., Лоскутов П. В., 2006
© Оформление, издательство "БХВ-Петербург", 2006

Оглавление

Предисловие **1**

Глава 1. Языки AutoLISP и Visual LISP **5**

1.1. Интерпретатор LISP 5
1.1.1. Вычисляемые выражения 6
1.1.2. Функция *setq* и точность отображения 8
1.1.3. Комментарии 9
1.2. Символы 9
1.2.1. Типы данных 10
1.2.2. Предопределенные символы 12
1.2.3. Переменные 12
1.2.4. Функция *load* 13
1.2.5. Функция *vl-load-all* 14
1.3. Выполнение команд AutoCAD 14
1.3.1. Функция *command* 14
1.3.2. Функция *vl-cmdf* 16
1.4. Арифметические функции 17
1.4.1. Функция + ... 17
1.4.2. Функция – 18
1.4.3. Функция * 18
1.4.4. Функция / 19
1.4.5. Функция *1+* 19
1.4.6. Функция *1–* 20
1.4.7. Функция ~ 20
1.4.8. Функция *abs* 20
1.5. Логические функции 21
1.5.1. Функция = ... 21
1.5.2. Функция /= 22
1.5.3. Функция < 22
1.5.4. Функция <= 23

1.5.5. Функция > ... 23
1.5.6. Функция >= ... 24
1.5.7. Функция *and* ... 24
1.5.8. Функция *or* ... 24
1.5.9. Функция *not* ... 25
1.5.10. Функция *null* ... 25
1.5.11. Функция *if* ... 26
1.5.12. Функция *progn* ... 26
1.5.13. Функция *cond* ... 27
1.5.14. Функция *minusp* ... 29
1.5.15. Функция *zerop* ... 30
1.5.16. Функция *numberp* ... 30
1.5.17. Функция *listp* ... 30
1.5.18. Функция *vl-consp* ... 31
1.5.19. Функция *atom* ... 31
1.5.20. Функция *vl-symbolp* ... 32
1.5.21. Функция *boundp* ... 32
1.5.22. Функция *eq* ... 32
1.5.23. Функция *equal* ... 33
1.5.24. Функция *while* ... 33
1.5.25. Функция *repeat* ... 35
1.6. Функции вычислений ... 35
1.6.1. Функция *fix* ... 35
1.6.2. Функция *rem* ... 36
1.6.3. Функция *max* ... 36
1.6.4. Функция *min* ... 37
1.6.5. Функция *gcd* ... 37
1.6.6. Функция *angle* ... 37
1.6.7. Функция *distance* ... 38
1.6.8. Функция *sqrt* ... 38
1.6.9. Функция *sin* ... 38
1.6.10. Функция *cos* ... 39
1.6.11. Функция *atan* ... 39
1.6.12. Функция *exp* ... 39
1.6.13. Функция *expt* ... 40
1.6.14. Функция *log* ... 40
1.6.15. Функция *logand* ... 40
1.6.16. Функция *logior* ... 41
1.6.17. Функция *lsh* ... 41
1.6.18. Функция *boole* ... 42
1.6.19. Функция *inters* ... 43
1.6.20. Функция *polar* ... 44
1.6.21. Функция *textbox* ... 44
1.6.22. Функция *vports* ... 44
1.6.23. Функция *vl-symbol-value* ... 47
1.6.24. Функция *set* ... 47

1.7. Функции преобразования данных ... 47
1.7.1. Функция *type* ... 47
1.7.2. Функция *float* ... 48
1.7.3. Функция *itoa* ... 49
1.7.4. Функция *rtos* ... 49
1.7.5. Функция *atoi* ... 50
1.7.6. Функция *atof* ... 51
1.7.7. Функция *angtof* ... 51
1.7.8. Функция *angtos* ... 52
1.7.9. Функция *distof* ... 52
1.7.10. Функция *trans* ... 53
1.7.11. Функция *cvunit* ... 55
1.7.12. Сообщения о неправильном типе ... 55
1.8. Функции обработки строк ... 56
1.8.1. Функция *chr* ... 56
1.8.2. Функция *ascii* ... 57
1.8.3. Функция *read* ... 57
1.8.4. Функция *strcase* ... 58
1.8.5. Функция *strcat* ... 59
1.8.6. Функция *strlen* ... 59
1.8.7. Функция *substr* ... 59
1.8.8. Функция *wcmatch* ... 60
1.8.9. Функция *vl-string->list* ... 61
1.8.10. Функция *vl-list->string* ... 62
1.8.11. Функция *vl-string-elt* ... 62
1.8.12. Функция *vl-string-left-trim* ... 62
1.8.13. Функция *vl-string-right-trim* ... 63
1.8.14. Функция *vl-string-trim* ... 63
1.8.15. Функция *vl-string-mismatch* ... 63
1.8.16. Функция *vl-string-position* ... 64
1.8.17. Функция *vl-string-search* ... 64
1.8.18. Функция *vl-string-subst* ... 65
1.8.19. Функция *vl-string-translate* ... 65
1.8.20. Функция *vl-symbol-name* ... 66
1.9. Функции обработки списков и точечных пар ... 66
1.9.1. Функция *list* ... 66
1.9.2. Функция *append* ... 67
1.9.3. Функция *nth* ... 67
1.9.4. Функция *reverse* ... 68
1.9.5. Функция *car* ... 68
1.9.6. Функция *cdr* ... 69
1.9.7. Функции *caar*, *cadr* и аналогичные им ... 69
1.9.8. Функция *cons* ... 70
1.9.9. Функция *vl-list** ... 71
1.9.10. Функция *member* ... 71

1.9.11. Функция *assoc* ... 72
1.9.12. Функция *apply* ... 72
1.9.13. Функция *mapcar* ... 73
1.9.14. Функция *vl-every* ... 73
1.9.15. Функция *foreach* ... 74
1.9.16. Функция *eval* ... 74
1.9.17. Функция *quote* ... 75
1.9.18. Функция *acad_strlsort* ... 75
1.9.19. Функция *last* ... 75
1.9.20. Функция *subst* ... 76
1.9.21. Функция *length* ... 76
1.9.22. Функция *vl-list-length* ... 76
1.9.23. Функция *vl-member-if* ... 77
1.9.24. Функция *vl-member-if-not* ... 77
1.9.25. Функция *vl-position* ... 78
1.9.26. Функция *vl-remove* ... 78
1.9.27. Функция *vl-remove-if* ... 78
1.9.28. Функция *vl-remove-if-not* ... 79
1.9.29. Функция *vl-some* ... 79
1.9.30. Функция *vl-sort* ... 80
1.9.31. Функция *vl-sort-i* ... 81
1.9.32. Функция *vl-catch-all-apply* и обработка ошибок ... 81
1.9.33. Функция *vl-catch-all-error-message* ... 82
1.9.34. Функция *vl-catch-all-error-p* ... 82
1.10. Функции ввода данных и указания объектов ... 83
1.10.1. Функция *initget* ... 83
1.10.2. Функция *getkword* ... 85
1.10.3. Функция *getint* ... 86
1.10.4. Функция *getreal* ... 87
1.10.5. Функция *getdist* ... 87
1.10.6. Функция *getangle* ... 88
1.10.7. Функция *getorient* ... 89
1.10.8. Функция *getpoint* ... 89
1.10.9. Функция *getcorner* ... 89
1.10.10. Функция *getstring* ... 90
1.10.11. Функция *getvar* ... 90
1.10.12. Функция *setvar* ... 91
1.10.13. Функция *getenv* ... 91
1.10.14. Функция *setenv* ... 92
1.10.15. Функция *getcfg* ... 93
1.10.16. Функция *setcfg* ... 93
1.10.17. Функция *getcname* ... 93
1.10.18. Функция *getfiled* ... 94
1.10.19. Функция *acad_colordlg* ... 96
1.10.20. Функция *acad_truecolordlg* ... 97
1.10.21. Функция *acad_truecolorcli* ... 99

1.11. Функции печати, вывода сообщений и доступа к файлам ... 99
1.11.1. Функция *findfile* ... 99
1.11.2. Функция *open* ... 100
1.11.3. Функция *close* ... 100
1.11.4. Функция *read-line* ... 100
1.11.5. Функция *write-line* ... 101
1.11.6. Функция *princ* ... 101
1.11.7. Функция *prin1* ... 102
1.11.8. Функция *print* ... 102
1.11.9. Функция *read-char* ... 102
1.11.10. Функция *write-char* ... 103
1.11.11. Функция *prompt* ... 103
1.11.12. Функция *alert* ... 103
1.11.13. Функция *terpri* ... 104
1.11.14. Функция *vl-directory-files* ... 104
1.11.15. Функция *vl-file-copy* ... 105
1.11.16. Функция *vl-file-delete* ... 106
1.11.17. Функция *vl-file-directory-p* ... 106
1.11.18. Функция *vl-file-rename* ... 107
1.11.19. Функция *vl-file-size* ... 107
1.11.20. Функция *vl-file-systime* ... 107
1.11.21. Функция *vl-filename-base* ... 108
1.11.22. Функция *vl-filename-directory* ... 108
1.11.23. Функция *vl-filename-extension* ... 108
1.11.24. Функция *vl-filename-mktemp* ... 109
1.11.25. Функция *vl-mkdir* ... 110
1.11.26. Функция *vl-princ-to-string* ... 110
1.11.27. Функция *vl-prin1-to-string* ... 111
1.11.28. Пример работы с файлами ... 111
1.12. Функции доступа к примитивам AutoCAD ... 112
1.12.1. Функция *entlast* ... 113
1.12.2. Функция *entnext* ... 114
1.12.3. Функция *entsel* ... 115
1.12.4. Функция *entdel* ... 115
1.12.5. Функция *entget* ... 116
1.12.6. Функция *entmake* ... 117
1.12.7. Функция *entmakex* ... 118
1.12.8. Функция *entmod* ... 118
1.12.9. Функция *entupd* ... 119
1.12.10. Функция *handent* ... 119
1.12.11. Функция *nentsel* ... 120
1.12.12. Функция *nentselp* ... 121
1.12.13. Наборы ... 121
1.12.14. Функция *ssget* ... 122
1.12.14.1. Фильтры со сложными условиями ... 124

1.12.15. Функция *ssadd* ... 126
1.12.16. Функция *ssdel* ... 126
1.12.17. Функция *sslength* ... 127
1.12.18. Функция *ssmemb* ... 127
1.12.19. Функция *ssname* ... 127
1.12.20. Функция *ssnamex* ... 128
1.12.21. Функция *sssetfirst* ... 130
1.12.22. Функция *ssgetfirst* ... 131
1.12.23. Пример работы с примитивами AutoCAD ... 131
1.13. Функции доступа к табличным данным ... 133
1.13.1. Символьные таблицы ... 133
1.13.2. Функция *tblsearch* ... 134
1.13.3. Функция *tblnext* ... 134
1.13.4. Функция *tblobjname* ... 135
1.13.5. Схема работы с таблицами ... 136
1.13.6. Функция *setview* ... 136
1.13.7. Функция *snvalid* ... 138
1.14. Функции, работающие с расширенными данными ... 139
1.14.1. Структура расширенных данных ... 139
1.14.2. Функция *regapp* ... 141
1.14.3. Присоединение расширенных данных ... 141
1.14.4. Функция *xdsize* ... 143
1.14.5. Функция *xdroom* ... 143
1.14.5.1. О примитиве VIEWPORT ... 143
1.15. Функции, работающие со словарями и X-записями ... 144
1.15.1. Словари ... 144
1.15.2. Функция *namedobjdict* ... 145
1.15.3. Функция *dictsearch* ... 146
1.15.4. Функция *dictnext* ... 147
1.15.5. Функция *dictadd* ... 149
1.15.6. Функция *dictremove* ... 150
1.15.7. Функция *dictrename* ... 151
1.15.8. Функция *layoutlist* ... 151
1.15.9. X-записи и наполнение словарей ... 152
1.16. Разработка функций пользователя ... 154
1.16.1. Функция *lambda* ... 154
1.16.2. Функция *defun* ... 155
1.16.3. Создание дополнительных команд системы AutoCAD с помощью пользовательских функций ... 158
1.16.4. Функция *defun-q* ... 159
1.16.5. Функция *defun-q-list-ref* ... 160
1.16.6. Функция *defun-q-list-set* ... 161
1.16.7. Функция *function* ... 161
1.16.8. Функция *vlisp-compile* ... 162
1.16.9. Функция *exit* ... 162

1.16.10. Функция *quit* ... 163
1.16.11. Функция **error** ... 163
1.16.12. Функция *vl-acad-defun* ... 164
1.16.13. Функция *vl-acad-undefun* ... 164
1.16.14. Функция *vl-exit-with-error* ... 165
1.16.15. Функция *vl-exit-with-value* ... 165
1.16.16. Функция *atoms-family* ... 165
1.16.17. Функция *trace* ... 166
1.16.18. Функция *untrace* ... 167
1.17. Функции работы с памятью, приложениями и реестром Windows ... 167
1.17.1. Функция *ver* ... 167
1.17.2. Функция *mem* ... 167
1.17.3. Функция *alloc* ... 168
1.17.4. Функция *expand* ... 168
1.17.5. Функция *gc* ... 169
1.17.6. Функция *vl-registry-read* ... 169
1.17.7. Функция *vl-registry-write* ... 169
1.17.8. Функция *vl-registry-delete* ... 170
1.17.9. Функция *vl-registry-descendents* ... 170
1.17.10. Функция *arx* ... 171
1.17.11. Функция *arxload* ... 171
1.17.12. Функция *arxunload* ... 171
1.17.13. Функция *autoload* ... 172
1.17.14. Функция *autoarxload* ... 172
1.17.15. Функция *vl-arx-import* ... 173
1.17.16. Функция *vl-doc-export* ... 173
1.17.17. Функция *vl-doc-import* ... 174
1.17.18. Функция *vl-list-exported-functions* ... 174
1.17.19. Функция *vl-list-loaded-vlx* ... 175
1.17.20. Функция *vl-vlx-loaded-p* ... 176
1.17.21. Функция *vl-unload-vlx* ... 176
1.17.22. Функция *vl-get-resource* ... 177
1.17.23. Функция *vl-vbaload* ... 177
1.17.24. Функция *vl-vbarun* ... 177
1.17.25. Функция *vl-bb-set* ... 178
1.17.26. Функция *vl-bb-ref* ... 178
1.17.27. Функция *vl-doc-set* ... 179
1.17.28. Функция *vl-doc-ref* ... 179
1.17.29. Функция *vl-load-all* ... 179
1.17.30. Функция *vl-propagate* ... 180
1.17.31. Глобальные и локальные переменные ... 180
1.17.32. Пространства имен ... 183
1.17.33. Проблемы многодокументного режима ... 184
1.17.34. Функция *startapp* ... 185

1.18. Другие функции ... 185
1.18.1. Функция *osnap* ... 185
1.18.2. Функция *redraw* ... 186
1.18.3. Функция *graphscr* ... 187
1.18.4. Функция *textscr* ... 187
1.18.5. Функция *textpage* ... 187
1.18.6. Функция *grtext* ... 187
1.18.7. Функция *grdraw* ... 190
1.18.8. Функция *grvecs* ... 190
1.18.9. Функция *grclear* ... 191
1.18.10. Функция *grread* ... 191
1.18.11. Функция *initdia* ... 194
1.18.12. Функция *tablet* ... 195
1.18.13. Функция *acad-push-dbmod* ... 196
1.18.14. Функция *acad-pop-dbmod* ... 196
1.18.15. Функция *acdimenableupdate* ... 196
1.18.16. Функция *acet-layerp-mode* ... 197
1.18.17. Функция *acet-layerp-mark* ... 197
1.18.18. Функция *acet-laytrans* ... 198
1.18.19. Функция *acet-ms-to-ps* ... 198
1.18.20. Функция *acet-ps-to-ms* ... 199
1.19. Файлы автоматической загрузки функций пользователя ... 199
1.19.1. Файл acad.lsp ... 200
1.19.2. Файл acaddoc.lsp ... 201
1.19.3. Системная переменная ACADLSPASDOC ... 201
1.19.4. Функция *s::startup* ... 201

Глава 2. Среда разработки приложений ... 202

2.1. Обзор меню ... 202
2.2. Панели инструментов Visual LISP ... 221
2.3. Написание и проверка синтаксиса программ в текстовом редакторе среды Visual LISP ... 222
2.4. Разработка приложений на языке LISP ... 223
2.4.1. Средства отладки программ ... 223
2.4.2. Проекты и FAS-приложения ... 225
2.4.3. VLX-приложения ... 228

Глава 3. Диалоговые окна ... 234

3.1. Язык DCL ... 235
3.1.1. Директивы ... 235
3.1.2. Пример диалогового окна ... 237
3.1.3. Директива описания диалога ... 238
3.2. Открытие и закрытие диалоговых окон ... 242
3.2.1. Функция *load_dialog* ... 242
3.2.2. Функция *unload_dialog* ... 242

3.2.3. Функция *new_dialog* ... 243
3.2.4. Функция *start_dialog* ... 243
3.2.5. Функция *done_dialog* ... 244
3.2.6. Функция *term_dialog* ... 244
3.2.7. Пример вызова диалогового окна ... 244
3.3. Элементы диалога ... 245
3.3.1. Типы элементов ... 245
3.3.2. Элементы с кнопкой выхода ... 246
3.3.3. Атрибуты элементов ... 247
3.3.4. Примеры элементов ... 251
3.3.4.1. Кнопка ... 251
3.3.4.2. Текстовые поля ... 252
3.3.4.3. Элементы с изображениями ... 252
3.3.4.4. Элементы со списками ... 253
3.3.4.5. Переключатели ... 254
3.3.4.6. Скользящая шкала ... 256
3.3.4.7. Флажок ... 257
3.3.5. Группировка полей ... 257
3.3.6. Элементы надписей ... 260
3.3.7. Элементы промежутков ... 261
3.4. Функции, работающие с элементами диалога ... 263
3.4.1. Функция *action_tile* ... 263
3.4.2. Функция *mode_tile* ... 264
3.4.3. Функция *set_tile* ... 265
3.4.4. Функция *get_tile* ... 265
3.4.5. Функция *get_attr* ... 265
3.4.6. Функция *client_data_tile* ... 266
3.4.7. Переменные действия ... 266
3.4.8. Пример ... 267
3.5. Функции, работающие со списками ... 272
3.5.1. Функция *start_list* ... 272
3.5.2. Функция *add_list* ... 273
3.5.3. Функция *end_list* ... 273
3.5.4. Пример окна со списками ... 273
3.5.5. Ограничения списков в DCL-файлах ... 274
3.5.6. Особенности применения функций *set_tile* и *get_tile* для списков ... 275
3.6. Функции, работающие с графическими элементами ... 276
3.6.1. Функция *start_image* ... 276
3.6.2. Функция *end_image* ... 277
3.6.3. Функция *fill_image* ... 277
3.6.4. Функция *slide_image* ... 278
3.6.5. Функция *vector_image* ... 278
3.6.6. Функция *dimx_tile* ... 279
3.6.7. Функция *dimy_tile* ... 279

3.6.8. Примеры окон с графическими элементами ... 279
3.6.8.1. Просмотр изображений из библиотеки слайдов ... 279
3.6.8.2. Изменение цветов графической кнопки ... 283
3.6.8.3. Изменение рисунка, наносимого на кнопку ... 285
3.7. Дополнительные приемы программирования диалоговых окон ... 289
3.7.1. Временный выход из диалогового окна ... 289
3.7.2. Вложенные диалоговые окна ... 291
3.7.3. Применение паролей ... 293

Глава 4. Меню и адаптации ... 294

4.1. Коротко о меню и адаптациях ... 294
4.1.1. Группы меню и группы адаптации ... 295
4.1.2. Разделы меню (адаптации) ... 295
4.2. Структура заголовков и пунктов меню ... 296
4.3. Функции LISP, работающие с меню ... 296
4.3.1. Функция *menugroup* ... 296
4.3.2. Функция *menucmd* ... 297
4.3.3. Операции загрузки (выгрузки) групп ... 298
4.4. Падающие меню ... 298
4.4.1. Добавление в строку меню ... 299
4.4.2. Удаление из строки меню ... 299
4.4.3. Вывод содержимого меню на экран ... 300
4.4.4. Отметка пункта меню ... 300
4.4.5. Отключение пункта меню ... 301
4.4.6. Восстановление пункта меню ... 301
4.4.7. Замена меню ... 301
4.4.8. Справка о состоянии пункта меню ... 302
4.5. Экранные меню ... 302
4.6. Графические меню ... 303
4.7. Панели инструментов ... 303
4.8. Контекстные меню ... 304
4.8.1. Разделы с контекстными меню ... 304
4.8.2. Схема работы контекстных меню ... 304
4.9. MNL-файлы ... 306
4.10. ActiveX ... 308

Глава 5. Справочные системы ... 309

5.1. HELP-файлы Windows ... 309
5.1.1. Структура HELP-файлов ... 309
5.1.2. CNT-файлы ... 311
5.1.3. Функция *help* ... 312
5.1.4. Вызов WIN-справки с помощью функции *startapp* ... 313
5.2. HTML-справки ... 313
5.2.1. Вызов HTML-справки с помощью функции *help* ... 313
5.2.2. Вызов HTML-справки с помощью функции *startapp* ... 313

5.3. Устаревшие справочные файлы AutoCAD ... 314
5.3.1. Функция *acad_helpdlg* ... 314
5.4. Контекстно-зависимая справка к командам пользователя ... 315
5.4.1. Функция *setfunhelp* ... 315
5.4.2. Особенности применения функции *setfunhelp* ... 315

Глава 6. Технология ActiveX ... 316

6.1. Расширение языка AutoLISP ... 316
6.1.1. Объекты и семейства ... 316
6.1.2. Функция *vl-load-com* ... 323
6.1.3. Свойства ... 324
6.1.4. Функция *vlax-get-acad-object* ... 324
6.1.5. Функция *vla-get-ActiveDocument* ... 324
6.1.6. Функция *vla-get-ModelSpace* ... 325
6.1.7. Функция *vla-get-PaperSpace* ... 325
6.1.8. Подготовка к работе с ActiveX ... 325
6.1.9. Методы ... 326
6.1.10. VLA-объекты ... 326
6.1.11. Функция *vlax-ename->vla-object* ... 327
6.1.12. Функция *vlax-vla-object->ename* ... 327
6.1.13. Работа со свойствами ... 327
6.1.14. Варианты ... 328
6.1.15. Функция *vlax-make-variant* ... 328
6.1.16. Функция *vlax-variant-type* ... 329
6.1.17. Функция *vlax-variant-value* ... 330
6.1.18. Функция *vlax-variant-change-type* ... 330
6.1.19. Функция *vlax-3D-point* ... 330
6.1.20. Безопасные массивы ... 331
6.1.21. Функция *vlax-make-safearray* ... 331
6.1.22. Функция *vlax-safearray-put-element* ... 332
6.1.23. Функция *vlax-safearray-fill* ... 332
6.1.24. Пример создания VLA-объекта ... 333
6.1.25. Функция *vlax-dump-object* ... 333
6.1.26. Другие функции ActiveX ... 335
6.2. Функции команд и методов ... 335
6.2.1. Функция *vlax-add-cmd* ... 335
6.2.2. Функция *vlax-remove-cmd* ... 336
6.2.3. Функция *vlax-invoke-method* ... 337
6.2.4. Функция *vlax-product-key* ... 337
6.3. Функции операций с объектами ... 337
6.3.1. Функция *vlax-erased-p* ... 338
6.3.2. Функция *vlax-method-applicable-p* ... 338
6.3.3. Функция *vlax-typeinfo-available-p* ... 338
6.3.4. Функция *vlax-object-released-p* ... 338

6.3.5. Функция *vlax-release-object* ... 339
6.3.6. Функция *vlax-read-enabled-p* ... 339
6.3.7. Функция *vlax-write-enabled-p* ... 339
6.3.8. Функция *vlax-create-object* ... 339
6.3.9. Функция *vlax-get-object* ... 340
6.3.10. Функция *vlax-get-or-create-object* ... 340
6.3.11. Функция *vlax-import-type-library* ... 340
6.3.12. Функция *vlax-map-collection* ... 341
6.3.13. Функция *vlax-for* ... 341
6.4. Функции операций со словарями ... 342
6.4.1. Функция *vlax-ldata-put* ... 342
6.4.2. Функция *vlax-ldata-get* ... 343
6.4.3. Функция *vlax-ldata-list* ... 343
6.4.4. Функция *vlax-ldata-delete* ... 343
6.4.5. Функция *vlax-ldata-test* ... 344
6.4.6. Словарь Extension Dictionary ... 344
6.4.6.1. Словарь фильтров слоев ... 344
6.4.6.2. Объект VLO-VL ... 346
6.5. Функции преобразований ... 348
6.5.1. Функция *vlax-safearray-get-dim* ... 348
6.5.2. Функция *vlax-safearray-get-l-bound* ... 348
6.5.3. Функция *vlax-safearray-get-u-bound* ... 348
6.5.4. Функция *vlax-safearray-get-element* ... 349
6.5.5. Функция *vlax-safearray-type* ... 349
6.5.6. Функция *vlax-safearray->list* ... 349
6.5.7. Функция *vlax-tmatrix* ... 349
6.6. Функции операций со свойствами ... 350
6.6.1. Функция *vlax-get-property* ... 350
6.6.2. Функция *vlax-put-property* ... 350
6.6.3. Функция *vlax-property-available-p* ... 350
6.7. Функции операций с кривыми ... 351
6.7.1. Функция *vlax-curve-getArea* ... 351
6.7.2. Функция *vlax-curve-getStartPoint* ... 351
6.7.3. Функция *vlax-curve-getEndPoint* ... 351
6.7.4. Функция *vlax-curve-getPointAtDist* ... 351
6.7.5. Функция *vlax-curve-getDistAtPoint* ... 352
6.7.6. Функция *vlax-curve-getStartParam* ... 352
6.7.7. Функция *vlax-curve-getEndParam* ... 352
6.7.8. Функция *vlax-curve-getParamAtPoint* ... 352
6.7.9. Функция *vlax-curve-getPointAtParam* ... 353
6.7.10. Функция *vlax-curve-getParamAtDist* ... 353
6.7.11. Функция *vlax-curve-getDistAtParam* ... 353
6.7.12. Функция *vlax-curve-isClosed* ... 353
6.7.13. Функция *vlax-curve-isPeriodic* ... 354
6.7.14. Функция *vlax-curve-isPlanar* ... 354

6.7.15. Функция *vlax-curve-getClosestPointTo* ... 354
6.7.16. Функция *vlax-curve-getClosestPointToProjection* ... 354
6.7.17. Функция *vlax-curve-getFirstDeriv* ... 355
6.7.18. Функция *vlax-curve-getSecondDeriv* ... 355
6.7.19. Что такое параметр кривой ... 355
6.8. Примеры работы с технологией ActiveX ... 355
6.8.1. Доступ к неактивному документу ... 356
6.8.2. Динамическое изменение контекстного меню ... 356

Глава 7. Методы ActiveX ... 359

7.1. Дополнительные соглашения ... 359
7.2. Функции методов ... 360
7.2.1. Метод *Activate* ... 362
7.2.2. Метод *Add* ... 362
7.2.3. Метод *Add3DFace* ... 365
7.2.4. Метод *Add3DMesh* ... 365
7.2.5. Метод *Add3DPoly* ... 366
7.2.6. Метод *AddArc* ... 366
7.2.7. Метод *AddAttribute* ... 367
7.2.8. Метод *AddBox* ... 367
7.2.9. Метод *AddCircle* ... 368
7.2.10. Метод *AddCone* ... 368
7.2.11. Метод *AddCustomInfo* ... 368
7.2.12. Метод *AddCustomObject* ... 369
7.2.13. Метод *AddCylinder* ... 369
7.2.14. Метод *AddDim3PointAngular* ... 369
7.2.15. Метод *AddDimAligned* ... 370
7.2.16. Метод *AddDimAngular* ... 370
7.2.17. Метод *AddDimArc* ... 371
7.2.18. Метод *AddDimDiametric* ... 371
7.2.19. Метод *AddDimOrdinate* ... 371
7.2.20. Метод *AddDimRadial* ... 372
7.2.21. Метод *AddDimRadialLarge* ... 372
7.2.22. Метод *AddDimRotated* ... 373
7.2.23. Метод *AddEllipse* ... 373
7.2.24. Метод *AddEllipticalCone* ... 374
7.2.25. Метод *AddEllipticalCylinder* ... 374
7.2.26. Метод *AddExtrudedSolid* ... 374
7.2.27. Метод *AddExtrudedSolidAlongPath* ... 375
7.2.28. Метод *AddFitPoint* ... 375
7.2.29. Метод *AddHatch* ... 376
7.2.30. Метод *AddItems* ... 377
7.2.31. Метод *AddLeader* ... 377
7.2.32. Метод *AddLightWeightPolyline* ... 378
7.2.33. Метод *AddLine* ... 378

7.2.34. Метод *AddMenuItem* ... 379
7.2.35. Метод *AddMInsertBlock* ... 379
7.2.36. Метод *AddMLine* ... 380
7.2.37. Метод *AddMText* ... 380
7.2.38. Метод *AddObject* ... 381
7.2.39. Метод *AddPoint* ... 381
7.2.40. Метод *AddPolyfaceMesh* ... 382
7.2.41. Метод *AddPolyline* ... 382
7.2.42. Метод *AddPViewport* ... 383
7.2.43. Метод *AddRaster* ... 383
7.2.44. Метод *AddRay* ... 384
7.2.45. Метод *AddRegion* ... 384
7.2.46. Метод *AddRevolvedSolid* ... 385
7.2.47. Метод *AddSeparator* ... 385
7.2.48. Метод *AddShape* ... 386
7.2.49. Метод *AddSolid* ... 386
7.2.50. Метод *AddSphere* ... 386
7.2.51. Метод *AddSpline* ... 387
7.2.52. Метод *AddSubMenu* ... 387
7.2.53. Метод *AddTable* ... 388
7.2.54. Метод *AddText* ... 388
7.2.55. Метод *AddTolerance* ... 389
7.2.56. Метод *AddToolbarButton* ... 389
7.2.57. Метод *AddTorus* ... 390
7.2.58. Метод *AddTrace* ... 390
7.2.59. Метод *AddVertex* ... 391
7.2.60. Метод *AddWedge* ... 391
7.2.61. Метод *AddXline* ... 391
7.2.62. Метод *AddXRecord* ... 392
7.2.63. Метод *AngleFromXAxis* ... 392
7.2.64. Метод *AngleToReal* ... 392
7.2.65. Метод *AngleToString* ... 393
7.2.66. Метод *AppendInnerLoop* ... 393
7.2.67. Метод *AppendItems* ... 394
7.2.68. Метод *AppendOuterLoop* ... 394
7.2.69. Метод *AppendVertex* ... 395
7.2.70. Метод *ArrayPolar* ... 395
7.2.71. Метод *ArrayRectangular* ... 396
7.2.72. Метод *AttachExternalReference* ... 396
7.2.73. Метод *AttachToolbarToFlyout* ... 397
7.2.74. Метод *AuditInfo* ... 398
7.2.75. Метод *Bind* ... 398
7.2.76. Метод *Block* ... 398
7.2.77. Метод *Boolean* ... 399
7.2.78. Метод *CheckInterference* ... 399

7.2.79. Метод *Clear* ... 399
7.2.80. Метод *ClearSubSelection* ... 400
7.2.81. Метод *ClearTableStyleOverrides* ... 400
7.2.82. Метод *ClipBoundary* ... 400
7.2.83. Метод *Close* ... 401
7.2.84. Метод *Connect* ... 402
7.2.85. Метод *ConvertToAnonymousBlock* ... 402
7.2.86. Метод *ConvertToStaticBlock* ... 403
7.2.87. Метод *Copy* ... 403
7.2.88. Метод *CopyFrom* ... 403
7.2.89. Метод *CopyObjects* ... 404
7.2.90. Метод *CopyProfile* ... 404
7.2.91. Метод *CreateEntry* ... 405
7.2.92. Метод *CreateLabel* ... 405
7.2.93. Метод *CreateLink* ... 406
7.2.94. Метод *CreateTypedArray* ... 407
7.2.95. Метод *Delete* ... 408
7.2.96. Метод *DeleteCellContent* ... 408
7.2.97. Метод *DeleteColumns* ... 408
7.2.98. Метод *DeleteConfiguration* ... 409
7.2.99. Метод *DeleteFitPoint* ... 409
7.2.100. Метод *DeleteProfile* ... 409
7.2.101. Метод *DeleteRows* ... 410
7.2.102. Метод *Detach* ... 410
7.2.103. Метод *Disconnect* ... 410
7.2.104. Метод *Display* ... 411
7.2.105. Метод *DisplayPlotPreview* ... 411
7.2.106. Метод *DistanceToReal* ... 411
7.2.107. Метод *Dock* ... 412
7.2.108. Метод *ElevateOrder* ... 412
7.2.109. Метод *EndUndoMark* ... 413
7.2.110. Метод *Erase* ... 413
7.2.111. Метод *Eval* ... 413
7.2.112. Метод *Evaluate* ... 414
7.2.113. Метод *Explode* ... 414
7.2.114. Метод *Export* ... 414
7.2.115. Метод *ExportProfile* ... 415
7.2.116. Метод *FieldCode* ... 416
7.2.117. Метод *Float* ... 416
7.2.118. Метод *GenerateLayout* ... 416
7.2.119. Метод *GenerateUsageData* ... 417
7.2.120. Метод *GetAcadState* ... 417
7.2.121. Метод *GetAlignment* ... 417
7.2.122. Метод *GetAllProfileNames* ... 418
7.2.123. Метод *GetAngle* ... 418

7.2.124. Метод *GetAttachmentPoint* ... 418
7.2.125. Метод *GetAttributes* ... 419
7.2.126. Метод *GetAutoScale* ... 419
7.2.127. Метод *GetBackgroundColor* ... 420
7.2.128. Метод *GetBackgroundColorNone* ... 420
7.2.129. Метод *GetBitmaps* ... 420
7.2.130. Метод *GetBlockAttributeValue* ... 421
7.2.131. Метод *GetBlockRotation* ... 421
7.2.132. Метод *GetBlockScale* ... 421
7.2.133. Метод *GetBlockTableRecordId* ... 422
7.2.134. Метод *GetBoundingBox* ... 422
7.2.135. Метод *GetBulge* ... 422
7.2.136. Метод *GetCanonicalMediaNames* ... 423
7.2.137. Метод *GetCellAlignment* ... 423
7.2.138. Метод *GetCellBackgroundColor* ... 423
7.2.139. Метод *GetCellBackgroundColorNone* ... 424
7.2.140. Метод *GetCellContentColor* ... 424
7.2.141. Метод *GetCellExtents* ... 424
7.2.142. Метод *GetCellGridColor* ... 425
7.2.143. Метод *GetCellGridLineWeight* ... 426
7.2.144. Метод *GetCellGridVisibility* ... 426
7.2.145. Метод *GetCellStyleOverrides* ... 426
7.2.146. Метод *GetCellTextHeight* ... 427
7.2.147. Метод *GetCellTextStyle* ... 427
7.2.148. Метод *GetCellType* ... 427
7.2.149. Метод *GetColor* ... 428
7.2.150. Метод *GetColumnWidth* ... 428
7.2.151. Метод *GetConstantAttributes* ... 428
7.2.152. Метод *GetContentColor* ... 429
7.2.153. Метод *GetControlPoint* ... 429
7.2.154. Метод *GetCorner* ... 430
7.2.155. Метод *GetCustomByIndex* ... 430
7.2.156. Метод *GetCustomByKey* ... 430
7.2.157. Метод *GetCustomScale* ... 431
7.2.158. Метод *GetDistance* ... 431
7.2.159. Метод *GetDynamicBlockProperties* ... 432
7.2.160. Метод *GetEntity* ... 432
7.2.161. Метод *GetExtensionDictionary* ... 432
7.2.162. Метод *GetFieldId* ... 433
7.2.163. Метод *GetFitPoint* ... 433
7.2.164. Метод *GetFont* ... 434
7.2.165. Метод *GetFullDrawOrder* ... 435
7.2.166. Метод *GetGridColor* ... 435
7.2.167. Метод *GetGridLineWeight* ... 436
7.2.168. Метод *GetGridSpacing* ... 436

7.2.169. Метод *GetGridVisibility* ... 437
7.2.170. Метод *GetInput* ... 437
7.2.171. Метод *GetInteger* ... 437
7.2.172. Метод *GetInterfaceObject* ... 438
7.2.173. Метод *GetInvisibleEdge* ... 438
7.2.174. Метод *GetKeyword* ... 438
7.2.175. Метод *GetLabelTemplates* ... 439
7.2.176. Метод *GetLinks* ... 439
7.2.177. Метод *GetLinksByKey* ... 440
7.2.178. Метод *GetLinksByLabelTemplate* ... 441
7.2.179. Метод *GetLinksBySQLCond* ... 441
7.2.180. Метод *GetLinkTemplates* ... 442
7.2.181. Метод *GetLocaleMediaName* ... 442
7.2.182. Метод *GetLoopAt* ... 443
7.2.183. Метод *GetMinimumColumnWidth* ... 443
7.2.184. Метод *GetMinimumRowHeight* ... 443
7.2.185. Метод *GetName* ... 444
7.2.186. Метод *GetObject* ... 444
7.2.187. Метод *GetOrientation* ... 444
7.2.188. Метод *GetPaperMargins* ... 445
7.2.189. Метод *GetPaperSize* ... 445
7.2.190. Метод *GetPlotDeviceNames* ... 446
7.2.191. Метод *GetPlotStyleTableNames* ... 446
7.2.192. Метод *GetPoint* ... 446
7.2.193. Метод *GetProjectFilePath* ... 447
7.2.194. Метод *GetQueries* ... 447
7.2.195. Метод *GetReal* ... 448
7.2.196. Метод *GetRelativeDrawOrder* ... 448
7.2.197. Метод *GetRemoteFile* ... 449
7.2.198. Метод *GetRowHeight* ... 449
7.2.199. Метод *GetRowType* ... 449
7.2.200. Метод *GetSnapSpacing* ... 450
7.2.201. Метод *GetString* ... 450
7.2.202. Метод *GetSubEntity* ... 451
7.2.203. Метод *GetSubSelection* ... 451
7.2.204. Метод *GetText* ... 452
7.2.205. Метод *GetTextHeight* ... 452
7.2.206. Метод *GetTextRotation* ... 453
7.2.207. Метод *GetTextStyle* ... 453
7.2.208. Метод *GetUCSMatrix* ... 453
7.2.209. Метод *GetVariable* ... 454
7.2.210. Метод *GetWeight* ... 454
7.2.211. Метод *GetWidth* ... 455
7.2.212. Метод *GetWindowToPlot* ... 455
7.2.213. Метод *GetXData* ... 455

7.2.214. Метод *GetXRecordData* ... 456
7.2.215. Метод *HandleToObject* ... 457
7.2.216. Метод *Hide* ... 457
7.2.217. Метод *Highlight* ... 457
7.2.218. Метод *HitTest* ... 458
7.2.219. Метод *Import* ... 458
7.2.220. Метод *ImportProfile* ... 459
7.2.221. Метод *IndexOf* ... 460
7.2.222. Метод *InitializeUserInput* ... 460
7.2.223. Метод *InsertBlock* ... 461
7.2.224. Метод *InsertColumns* ... 461
7.2.225. Метод *InsertInMenuBar* ... 462
7.2.226. Метод *InsertLoopAt* ... 462
7.2.227. Метод *InsertMenuInMenuBar* ... 463
7.2.228. Метод *InsertRows* ... 463
7.2.229. Метод *Intersect* ... 464
7.2.230. Метод *IntersectWith* ... 464
7.2.231. Метод *IsConnected* ... 465
7.2.232. Метод *IsMergedCell* ... 465
7.2.233. Метод *IsRemoteFile* ... 466
7.2.234. Метод *IsURL* ... 466
7.2.235. Метод *Item* ... 467
7.2.236. Метод *LaunchBrowserDialog* ... 467
7.2.237. Метод *ListArx* ... 468
7.2.238. Метод *Load* ... 468
7.2.239. Метод *LoadArx* ... 469
7.2.240. Метод *LoadDVB* ... 469
7.2.241. Метод *LoadShapeFile* ... 469
7.2.242. Метод *MergeCells* ... 470
7.2.243. Метод *Mirror* ... 470
7.2.244. Метод *Mirror3D* ... 470
7.2.245. Метод *Move* ... 471
7.2.246. Метод *MoveAbove* ... 471
7.2.247. Метод *MoveBelow* ... 472
7.2.248. Метод *MoveToBottom* ... 472
7.2.249. Метод *MoveToTop* ... 473
7.2.250. Метод *New* ... 473
7.2.251. Метод *NumCustomInfo* ... 473
7.2.252. Метод *ObjectIDToObject* ... 474
7.2.253. Метод *Offset* ... 474
7.2.254. Метод *Open* ... 475
7.2.255. Метод *PlotToDevice* ... 475
7.2.256. Метод *PlotToFile* ... 475
7.2.257. Метод *PolarPoint* ... 476
7.2.258. Метод *Prompt* ... 476

7.2.259. Метод *PurgeAll* ... 477
7.2.260. Метод *PurgeFitData* ... 477
7.2.261. Метод *PutRemoteFile* ... 477
7.2.262. Метод *Quit* ... 477
7.2.263. Метод *RealToString* ... 478
7.2.264. Метод *RecomputeTableBlock* ... 478
7.2.265. Метод *Refresh* ... 478
7.2.266. Метод *RefreshPlotDeviceInfo* ... 479
7.2.267. Метод *Regen* ... 479
7.2.268. Метод *Reload* ... 480
7.2.269. Метод *ReloadLabels* ... 480
7.2.270. Метод *Remove* ... 481
7.2.271. Метод *RemoveCustomByIndex* ... 481
7.2.272. Метод *RemoveCustomByKey* ... 481
7.2.273. Метод *RemoveEntry* ... 481
7.2.274. Метод *RemoveFromMenuBar* ... 482
7.2.275. Метод *RemoveItems* ... 482
7.2.276. Метод *RemoveMenuFromMenuBar* ... 483
7.2.277. Метод *Rename* ... 483
7.2.278. Метод *RenameProfile* ... 483
7.2.279. Метод *Replace* ... 483
7.2.280. Метод *ReselectSubRegion* ... 484
7.2.281. Метод *ResetBlock* ... 484
7.2.282. Метод *ResetProfile* ... 484
7.2.283. Метод *Restore* ... 485
7.2.284. Метод *Reverse* ... 485
7.2.285. Метод *Rotate* ... 485
7.2.286. Метод *Rotate3D* ... 486
7.2.287. Метод *RunMacro* ... 486
7.2.288. Метод *Save* ... 487
7.2.289. Метод *SaveAs* ... 487
7.2.290. Метод *ScaleEntity* ... 488
7.2.291. Метод *SectionSolid* ... 488
7.2.292. Метод *Select* ... 489
7.2.293. Метод *SelectAtPoint* ... 490
7.2.294. Метод *SelectByPolygon* ... 490
7.2.295. Метод *SelectOnScreen* ... 491
7.2.296. Метод *SelectSubRegion* ... 492
7.2.297. Метод *SendCommand* ... 492
7.2.298. Метод *SendModelessOperationEnded* ... 493
7.2.299. Метод *SendModelessOperationStart* ... 493
7.2.300. Метод *SetAlignment* ... 493
7.2.301. Метод *SetAutoScale* ... 494
7.2.302. Метод *SetBackgroundColor* ... 494
7.2.303. Метод *SetBackgroundColorNone* ... 495

7.2.304. Метод *SetBitmaps* ... 495
7.2.305. Метод *SetBlockAttributeValue* ... 495
7.2.306. Метод *SetBlockRotation* ... 496
7.2.307. Метод *SetBlockScale* ... 496
7.2.308. Метод *SetBlockTableRecordId* ... 496
7.2.309. Метод *SetBulge* ... 497
7.2.310. Метод *SetCellAlignment* ... 497
7.2.311. Метод *SetCellBackgroundColor* ... 498
7.2.312. Метод *SetCellBackgroundColorNone* ... 498
7.2.313. Метод *SetCellContentColor* ... 498
7.2.314. Метод *SetCellGridColor* ... 499
7.2.315. Метод *SetCellGridLineWeight* ... 499
7.2.316. Метод *SetCellGridVisibility* ... 500
7.2.317. Метод *SetCellTextHeight* ... 500
7.2.318. Метод *SetCellTextStyle* ... 500
7.2.319. Метод *SetCellType* ... 501
7.2.320. Метод *SetColor* ... 501
7.2.321. Метод *SetColorBookColor* ... 501
7.2.322. Метод *SetColumnWidth* ... 502
7.2.323. Метод *SetContentColor* ... 502
7.2.324. Метод *SetControlPoint* ... 503
7.2.325. Метод *SetCustomByIndex* ... 503
7.2.326. Метод *SetCustomByKey* ... 503
7.2.327. Метод *SetCustomScale* ... 504
7.2.328. Метод *SetDatabase* ... 504
7.2.329. Метод *SetFieldId* ... 504
7.2.330. Метод *SetFitPoint* ... 505
7.2.331. Метод *SetFont* ... 505
7.2.332. Метод *SetGridColor* ... 505
7.2.333. Метод *SetGridLineWeight* ... 506
7.2.334. Метод *SetGridSpacing* ... 506
7.2.335. Метод *SetGridVisibility* ... 506
7.2.336. Метод *SetInvisibleEdge* ... 507
7.2.337. Метод *SetLayoutsToPlot* ... 507
7.2.338. Метод *SetNames* ... 508
7.2.339. Метод *SetPattern* ... 508
7.2.340. Метод *SetProjectFilePath* ... 508
7.2.341. Метод *SetRelativeDrawOrder* ... 509
7.2.342. Метод *SetRGB* ... 509
7.2.343. Метод *SetRowHeight* ... 509
7.2.344. Метод *SetSnapSpacing* ... 510
7.2.345. Метод *SetSubSelection* ... 510
7.2.346. Метод *SetText* ... 510
7.2.347. Метод *SetTextHeight* ... 511
7.2.348. Метод *SetTextRotation* ... 511

7.2.349. Метод *SetTextStyle* ... 511
7.2.350. Метод *SetVariable* ... 512
7.2.351. Метод *SetView* ... 512
7.2.352. Метод *SetWeight* ... 512
7.2.353. Метод *SetWidth* ... 513
7.2.354. Метод *SetWindowToPlot* ... 513
7.2.355. Метод *SetXData* ... 513
7.2.356. Метод *SetXRecordData* ... 514
7.2.357. Метод *Show* ... 515
7.2.358. Метод *SliceSolid* ... 515
7.2.359. Метод *Split* ... 516
7.2.360. Метод *StartBatchMode* ... 516
7.2.361. Метод *StartUndoMark* ... 517
7.2.362. Метод *Subtract* ... 517
7.2.363. Метод *SwapOrder* ... 517
7.2.364. Метод *SyncModelView* ... 518
7.2.365. Метод *TransformBy* ... 518
7.2.366. Метод *TranslateCoordinates* ... 518
7.2.367. Метод *Union* ... 519
7.2.368. Метод *Unload* ... 520
7.2.369. Метод *UnloadArx* ... 520
7.2.370. Метод *UnloadDVB* ... 521
7.2.371. Метод *UnmergeCells* ... 521
7.2.372. Метод *Update* ... 521
7.2.373. Метод *UpdateEntry* ... 521
7.2.374. Метод *Wblock* ... 522
7.2.375. Метод *ZoomAll* ... 522
7.2.376. Метод *ZoomCenter* ... 522
7.2.377. Метод *ZoomExtents* ... 523
7.2.378. Метод *ZoomPickWindow* ... 523
7.2.379. Метод *ZoomPrevious* ... 523
7.2.380. Метод *ZoomScaled* ... 523
7.2.381. Метод *ZoomWindow* ... 524

Глава 8. Свойства ActiveX ... 525

8.1. Дополнительные соглашения ... 525
8.2. Функции свойств ... 525
8.2.1. Свойство *Action* ... 527
8.2.2. Свойство *Active* ... 528
8.2.3. Свойство *ActiveDimStyle* ... 528
8.2.4. Свойство *ActiveDocument* ... 528
8.2.5. Свойство *ActiveLayer* ... 529
8.2.6. Свойство *ActiveLayout* ... 529
8.2.7. Свойство *ActiveLinetype* ... 529
8.2.8. Свойство *ActiveProfile* ... 529
8.2.9. Свойство *ActivePViewport* ... 530

8.2.10. Свойство *ActiveSelectionSet* ... 530
8.2.11. Свойство *ActiveSpace* ... 530
8.2.12. Свойство *ActiveTextStyle* ... 530
8.2.13. Свойство *ActiveUCS* ... 531
8.2.14. Свойство *ActiveViewport* ... 531
8.2.15. Свойство *ADCInsertUnitsDefaultSource* ... 531
8.2.16. Свойство *ADCInsertUnitsDefaultTarget* ... 532
8.2.17. Свойство *AffectsGraphics* ... 532
8.2.18. Свойство *Algorithm* ... 533
8.2.19. Свойство *Alignment* ... 533
8.2.20. Свойство *AlignmentPointAcquisition* ... 534
8.2.21. Свойство *AllowedValues* ... 534
8.2.22. Свойство *AllowLongSymbolNames* ... 535
8.2.23. Свойство *AltFontFile* ... 535
8.2.24. Свойство *AltRoundDistance* ... 535
8.2.25. Свойство *AltSuppressLeadingZeros* ... 536
8.2.26. Свойство *AltSuppressTrailingZeros* ... 536
8.2.27. Свойство *AltSuppressZeroFeet* ... 536
8.2.28. Свойство *AltSuppressZeroInches* ... 537
8.2.29. Свойство *AltTabletMenuFile* ... 537
8.2.30. Свойство *AltTextPrefix* ... 537
8.2.31. Свойство *AltTextSuffix* ... 538
8.2.32. Свойство *AltTolerancePrecision* ... 538
8.2.33. Свойство *AltToleranceSuppressLeadingZeros* ... 538
8.2.34. Свойство *AltToleranceSuppressTrailingZeros* ... 539
8.2.35. Свойство *AltToleranceSuppressZeroFeet* ... 539
8.2.36. Свойство *AltToleranceSuppressZeroInches* ... 540
8.2.37. Свойство *AltUnits* ... 540
8.2.38. Свойство *AltUnitsFormat* ... 540
8.2.39. Свойство *AltUnitsPrecision* ... 541
8.2.40. Свойство *AltUnitsScale* ... 541
8.2.41. Свойство *Angle* ... 541
8.2.42. Свойство *AngleFormat* ... 542
8.2.43. Свойство *AngleVertex* ... 542
8.2.44. Свойство *Annotation* ... 542
8.2.45. Свойство *Application* ... 543
8.2.46. Свойство *ArcEndParam* ... 543
8.2.47. Свойство *ArcLength* ... 543
8.2.48. Свойство *ArcPoint* ... 544
8.2.49. Свойство *ArcSmoothness* ... 544
8.2.50. Свойство *ArcStartParam* ... 544
8.2.51. Свойство *Area* ... 545
8.2.52. Свойство *Arrowhead1Block* ... 545
8.2.53. Свойство *Arrowhead1Type* ... 545
8.2.54. Свойство *Arrowhead2Block* ... 546

8.2.55. Свойство *Arrowhead2Type* ... 547
8.2.56. Свойство *ArrowheadBlock* ... 547
8.2.57. Свойство *ArrowheadSize* ... 547
8.2.58. Свойство *ArrowheadType* ... 548
8.2.59. Свойство *AssociativeHatch* ... 548
8.2.60. Свойство *AttachmentPoint* ... 548
8.2.61. Свойство *Author* ... 549
8.2.62. Свойство *AutoAudit* ... 549
8.2.63. Свойство *AutomaticPlotLog* ... 549
8.2.64. Свойство *AutoSaveInterval* ... 550
8.2.65. Свойство *AutoSavePath* ... 550
8.2.66. Свойство *AutoSnapAperture* ... 550
8.2.67. Свойство *AutoSnapApertureSize* ... 551
8.2.68. Свойство *AutoSnapMagnet* ... 551
8.2.69. Свойство *AutoSnapMarker* ... 551
8.2.70. Свойство *AutoSnapMarkerColor* ... 552
8.2.71. Свойство *AutoSnapMarkerSize* ... 552
8.2.72. Свойство *AutoSnapToolTip* ... 553
8.2.73. Свойство *AutoTrackingVecColor* ... 553
8.2.74. Свойство *AutoTrackTooltip* ... 553
8.2.75. Свойство *BackgroundFill* ... 554
8.2.76. Свойство *Backward* ... 554
8.2.77. Свойство *BasePoint* ... 554
8.2.78. Свойство *BatchPlotProgress* ... 555
8.2.79. Свойство *BeepOnError* ... 555
8.2.80. Свойство *BigFontFile* ... 555
8.2.81. Свойство *BitFlags* ... 556
8.2.82. Свойство *Block* ... 556
8.2.83. Свойство *Blocks* ... 556
8.2.84. Свойство *BlockScaling* ... 557
8.2.85. Свойство *Blue* ... 557
8.2.86. Свойство *BookName* ... 557
8.2.87. Свойство *Brightness* ... 557
8.2.88. Свойство *CanonicalMediaName* ... 558
8.2.89. Свойство *Caption* ... 558
8.2.90. Свойство *Catalog* ... 558
8.2.91. Свойство *CategoryName* ... 559
8.2.92. Свойство *Center* ... 559
8.2.93. Свойство *CenterMarkSize* ... 559
8.2.94. Свойство *CenterPlot* ... 560
8.2.95. Свойство *CenterPoint* ... 560
8.2.96. Свойство *CenterType* ... 560
8.2.97. Свойство *Centroid* ... 561
8.2.98. Свойство *Check* ... 561
8.2.99. Свойство *ChordPoint* ... 562

8.2.100. Свойство *Circumference* ... 562
8.2.101. Свойство *Clipped* ... 562
8.2.102. Свойство *ClippingEnabled* ... 563
8.2.103. Свойство *Closed* ... 563
8.2.104. Свойство *Color* ... 563
8.2.105. Свойство *ColorBookPath* ... 564
8.2.106. Свойство *ColorIndex* ... 564
8.2.107. Свойство *ColorMethod* ... 564
8.2.108. Свойство *ColorName* ... 565
8.2.109. Свойство *Columns* ... 565
8.2.110. Свойство *ColumnSpacing* ... 565
8.2.111. Свойство *ColumnWidth* ... 566
8.2.112. Свойство *Comment* ... 566
8.2.113. Свойство *Comments* ... 566
8.2.114. Свойство *ConfigFile* ... 566
8.2.115. Свойство *ConfigName* ... 567
8.2.116. Свойство *Constant* ... 567
8.2.117. Свойство *ConstantWidth* ... 567
8.2.118. Свойство *ContinuousPlotLog* ... 567
8.2.119. Свойство *ContourLinesPerSurface* ... 568
8.2.120. Свойство *Contrast* ... 568
8.2.121. Свойство *ControlPoints* ... 568
8.2.122. Свойство *Coordinate* ... 569
8.2.123. Свойство *Coordinates* ... 569
8.2.124. Свойство *Count* ... 570
8.2.125. Свойство *CreateBackup* ... 570
8.2.126. Свойство *CursorSize* ... 570
8.2.127. Свойство *CustomDictionary* ... 571
8.2.128. Свойство *CustomScale* ... 571
8.2.129. Свойство *Database* ... 571
8.2.130. Свойство *DataSource* ... 571
8.2.131. Свойство *DataSourceLocation* ... 572
8.2.132. Свойство *DbConnect* ... 572
8.2.133. Свойство *DecimalSeparator* ... 572
8.2.134. Свойство *DefaultInternetURL* ... 573
8.2.135. Свойство *DefaultOutputDevice* ... 573
8.2.136. Свойство *DefaultPlotStyleForLayer* ... 573
8.2.137. Свойство *DefaultPlotStyleForObjects* ... 574
8.2.138. Свойство *DefaultPlotStyleTable* ... 574
8.2.139. Свойство *DefaultPlotToFilePath* ... 574
8.2.140. Свойство *DefinedSize* ... 575
8.2.141. Свойство *Degree* ... 575
8.2.142. Свойство *Delta* ... 575
8.2.143. Свойство *DemandLoadARXApp* ... 576
8.2.144. Свойство *Description* ... 576

8.2.145. Свойство *Diameter* ... 576
8.2.146. Свойство *Dictionaries* ... 577
8.2.147. Свойство *DimensionLineColor* ... 577
8.2.148. Свойство *DimensionLineExtend* ... 577
8.2.149. Свойство *DimensionLineWeight* ... 578
8.2.150. Свойство *DimLine1Suppress* ... 579
8.2.151. Свойство *DimLine2Suppress* ... 579
8.2.152. Свойство *DimLineInside* ... 579
8.2.153. Свойство *DimLineSuppress* ... 580
8.2.154. Свойство *DimStyles* ... 580
8.2.155. Свойство *Direction* ... 580
8.2.156. Свойство *DirectionVector* ... 581
8.2.157. Свойство *Display* ... 581
8.2.158. Свойство *DisplayableAttributes* ... 582
8.2.159. Свойство *DisplayGrips* ... 582
8.2.160. Свойство *DisplayGripsWithinBlocks* ... 582
8.2.161. Свойство *DisplayLayoutTabs* ... 583
8.2.162. Свойство *DisplayLocked* ... 583
8.2.163. Свойство *DisplayOLEScale* ... 583
8.2.164. Свойство *DisplayScreenMenu* ... 584
8.2.165. Свойство *DisplayScrollBars* ... 584
8.2.166. Свойство *DisplaySilhouette* ... 584
8.2.167. Свойство *DisplayValues* ... 585
8.2.168. Свойство *DockedVisibleLines* ... 585
8.2.169. Свойство *DockStatus* ... 586
8.2.170. Свойство *Document* ... 586
8.2.171. Свойство *Documents* ... 586
8.2.172. Свойство *Drafting* ... 587
8.2.173. Свойство *DrawingDirection* ... 587
8.2.174. Свойство *DriversPath* ... 587
8.2.175. Свойство *EffectiveName* ... 588
8.2.176. Свойство *Elevation* ... 588
8.2.177. Свойство *ElevationModelSpace* ... 588
8.2.178. Свойство *ElevationPaperSpace* ... 589
8.2.179. Свойство *Enable* ... 589
8.2.180. Свойство *EnableStartupDialog* ... 589
8.2.181. Свойство *EndAngle* ... 590
8.2.182. Свойство *EndParameter* ... 590
8.2.183. Свойство *EndPoint* ... 590
8.2.184. Свойство *EndSubMenuLevel* ... 591
8.2.185. Свойство *EndTangent* ... 591
8.2.186. Свойство *EntityColor* ... 591
8.2.187. Свойство *Explodable* ... 592
8.2.188. Свойство *ExtensionLineColor* ... 592
8.2.189. Свойство *ExtensionLineExtend* ... 592

8.2.190. Свойство *ExtensionLineOffset* ... 593
8.2.191. Свойство *ExtensionLineWeight* ... 593
8.2.192. Свойство *ExtLine1EndPoint* ... 593
8.2.193. Свойство *ExtLine1Point* ... 594
8.2.194. Свойство *ExtLine1StartPoint* ... 594
8.2.195. Свойство *ExtLine1Suppress* ... 594
8.2.196. Свойство *ExtLine2EndPoint* ... 595
8.2.197. Свойство *ExtLine2Point* ... 595
8.2.198. Свойство *ExtLine2StartPoint* ... 595
8.2.199. Свойство *ExtLine2Suppress* ... 596
8.2.200. Свойство *Fade* ... 596
8.2.201. Свойство *Feature* ... 596
8.2.202. Свойство *FieldLength* ... 596
8.2.203. Свойство *FieldName* ... 597
8.2.204. Свойство *FileDependencies* ... 597
8.2.205. Свойство *FileName* ... 597
8.2.206. Свойство *Files* ... 598
8.2.207. Свойство *FileSize* ... 598
8.2.208. Свойство *FingerprintGuid* ... 598
8.2.209. Свойство *Fit* ... 598
8.2.210. Свойство *FitPoints* ... 599
8.2.211. Свойство *FitTolerance* ... 599
8.2.212. Свойство *FloatingRows* ... 600
8.2.213. Свойство *FlowDirection* ... 600
8.2.214. Свойство *Flyout* ... 600
8.2.215. Свойство *FontFile* ... 601
8.2.216. Свойство *FontFileMap* ... 601
8.2.217. Свойство *ForceLineInside* ... 601
8.2.218. Свойство *Format* ... 602
8.2.219. Свойство *FoundPath* ... 602
8.2.220. Свойство *FractionFormat* ... 602
8.2.221. Свойство *Freeze* ... 603
8.2.222. Свойство *FullCRCValidation* ... 603
8.2.223. Свойство *FullFileName* ... 604
8.2.224. Свойство *FullName* ... 604
8.2.225. Свойство *FullScreenTrackingVector* ... 605
8.2.226. Свойство *GradientAngle* ... 605
8.2.227. Свойство *GradientCentered* ... 605
8.2.228. Свойство *GradientColor1* ... 606
8.2.229. Свойство *GradientColor2* ... 606
8.2.230. Свойство *GradientName* ... 607
8.2.231. Свойство *GraphicsWinLayoutBackgrndColor* ... 607
8.2.232. Свойство *GraphicsWinModelBackgrndColor* ... 608
8.2.233. Свойство *Green* ... 608
8.2.234. Свойство *GridOn* ... 608

8.2.235. Свойство *GripColorSelected* ... 609
8.2.236. Свойство *GripColorUnselected* ... 609
8.2.237. Свойство *GripSize* ... 610
8.2.238. Свойство *Groups* ... 610
8.2.239. Свойство *Handle* ... 610
8.2.240. Свойство *HasAttributes* ... 610
8.2.241. Свойство *HasExtensionDictionary* ... 611
8.2.242. Свойство *HasLeader* ... 611
8.2.243. Свойство *HasSheetView* ... 611
8.2.244. Свойство *HasSubSelection* ... 612
8.2.245. Свойство *HasVpAssociation* ... 612
8.2.246. Свойство *HatchObjectType* ... 612
8.2.247. Свойство *HatchStyle* ... 613
8.2.248. Свойство *HeaderSuppressed* ... 613
8.2.249. Свойство *Height* ... 613
8.2.250. Свойство *HelpFilePath* ... 614
8.2.251. Свойство *HelpString* ... 614
8.2.252. Свойство *HistoryLines* ... 614
8.2.253. Свойство *HorizontalTextPosition* ... 615
8.2.254. Свойство *HorzCellMargin* ... 615
8.2.255. Свойство *HWND* ... 615
8.2.256. Свойство *HyperlinkBase* ... 616
8.2.257. Свойство *HyperlinkDisplayCursor* ... 616
8.2.258. Свойство *HyperlinkDisplayTooltip* ... 616
8.2.259. Свойство *Hyperlinks* ... 617
8.2.260. Свойство *ImageFile* ... 617
8.2.261. Свойство *ImageFrameHighlight* ... 617
8.2.262. Свойство *ImageHeight* ... 617
8.2.263. Свойство *ImageVisibility* ... 618
8.2.264. Свойство *ImageWidth* ... 618
8.2.265. Свойство *IncrementalSavePercent* ... 619
8.2.266. Свойство *Index* ... 619
8.2.267. Свойство *InsertionPoint* ... 619
8.2.268. Свойство *InsUnits* ... 620
8.2.269. Свойство *InsUnitsFactor* ... 620
8.2.270. Свойство *Invisible* ... 620
8.2.271. Свойство *IsCloned* ... 620
8.2.272. Свойство *IsDynamicBlock* ... 621
8.2.273. Свойство *IsLayout* ... 621
8.2.274. Свойство *IsModified* ... 622
8.2.275. Свойство *ISOPenWidth* ... 622
8.2.276. Свойство *IsOwnerXlated* ... 623
8.2.277. Свойство *IsPartial* ... 623
8.2.278. Свойство *IsPeriodic* ... 624
8.2.279. Свойство *IsPlanar* ... 624

8.2.280. Свойство *IsPrimary* ... 624
8.2.281. Свойство *IsQuiescent* ... 625
8.2.282. Свойство *IsRational* ... 625
8.2.283. Свойство *Issuer* ... 625
8.2.284. Свойство *IsXRef* ... 626
8.2.285. Свойство *JogAngle* ... 626
8.2.286. Свойство *JogLocation* ... 626
8.2.287. Свойство *Justification* ... 627
8.2.288. Свойство *Key* ... 627
8.2.289. Свойство *KeyboardAccelerator* ... 628
8.2.290. Свойство *KeyboardPriority* ... 628
8.2.291. Свойство *KeyDescriptions* ... 628
8.2.292. Свойство *KeyLength* ... 629
8.2.293. Свойство *KeyValues* ... 629
8.2.294. Свойство *Keywords* ... 629
8.2.295. Свойство *Knots* ... 630
8.2.296. Свойство *Label* ... 630
8.2.297. Свойство *LabelBlockId* ... 631
8.2.298. Свойство *LabelTemplate* ... 631
8.2.299. Свойство *LargeButtons* ... 632
8.2.300. Свойство *LastHeight* ... 632
8.2.301. Свойство *LastSavedBy* ... 632
8.2.302. Свойство *Layer* ... 633
8.2.303. Свойство *LayerOn* ... 633
8.2.304. Свойство *Layers* ... 633
8.2.305. Свойство *LayerState* ... 633
8.2.306. Свойство *Layout* ... 634
8.2.307. Свойство *LayoutCreateViewport* ... 634
8.2.308. Свойство *LayoutCrosshairColor* ... 634
8.2.309. Свойство *LayoutDisplayMargins* ... 635
8.2.310. Свойство *LayoutDisplayPaper* ... 635
8.2.311. Свойство *LayoutDisplayPaperShadow* ... 635
8.2.312. Свойство *LayoutId* ... 636
8.2.313. Свойство *Layouts* ... 636
8.2.314. Свойство *LayoutShowPlotSetup* ... 636
8.2.315. Свойство *Leader1Point* ... 637
8.2.316. Свойство *Leader2Point* ... 637
8.2.317. Свойство *LeaderAttachment* ... 637
8.2.318. Свойство *LeaderLength* ... 638
8.2.319. Свойство *LeaderObjectId* ... 638
8.2.320. Свойство *LeaderOffset* ... 638
8.2.321. Свойство *Left* ... 639
8.2.322. Свойство *Length* ... 639
8.2.323. Свойство *LensLength* ... 639
8.2.324. Свойство *Limits* ... 640

8.2.325. Свойство *LinearScaleFactor* ... 640
8.2.326. Свойство *LineSpacingDistance* ... 641
8.2.327. Свойство *LineSpacingFactor* ... 641
8.2.328. Свойство *LineSpacingStyle* ... 641
8.2.329. Свойство *Linetype* ... 642
8.2.330. Свойство *LinetypeGeneration* ... 642
8.2.331. Свойство *Linetypes* ... 642
8.2.332. Свойство *LinetypeScale* ... 642
8.2.333. Свойство *LineWeight* ... 643
8.2.334. Свойство *LineWeightDisplay* ... 643
8.2.335. Свойство *Link* ... 643
8.2.336. Свойство *LinkTemplate* ... 644
8.2.337. Свойство *LinkType* ... 644
8.2.338. Свойство *LoadAcadLspInAllDocuments* ... 645
8.2.339. Свойство *LocaleId* ... 645
8.2.340. Свойство *Lock* ... 645
8.2.341. Свойство *LockAspectRatio* ... 646
8.2.342. Свойство *LockPosition* ... 646
8.2.343. Свойство *LogFileOn* ... 646
8.2.344. Свойство *LogFilePath* ... 647
8.2.345. Свойство *LowerLeftCorner* ... 647
8.2.346. Свойство *Macro* ... 647
8.2.347. Свойство *MainDictionary* ... 648
8.2.348. Свойство *MajorAxis* ... 648
8.2.349. Свойство *MajorRadius* ... 648
8.2.350. Свойство *Mask* ... 649
8.2.351. Свойство *MaxActiveViewports* ... 650
8.2.352. Свойство *MaxAutoCADWindow* ... 650
8.2.353. Свойство *MClose* ... 650
8.2.354. Свойство *MDensity* ... 651
8.2.355. Свойство *Measurement* ... 651
8.2.356. Свойство *MenuBar* ... 651
8.2.357. Свойство *MenuFile* ... 651
8.2.358. Свойство *MenuFileName* ... 652
8.2.359. Свойство *MenuGroups* ... 652
8.2.360. Свойство *Menus* ... 652
8.2.361. Свойство *MinimumTableHeight* ... 653
8.2.362. Свойство *MinimumTableWidth* ... 653
8.2.363. Свойство *MinorAxis* ... 653
8.2.364. Свойство *MinorRadius* ... 653
8.2.365. Свойство *MLineScale* ... 654
8.2.366. Свойство *Mode* ... 654
8.2.367. Свойство *ModelCrosshairColor* ... 655
8.2.368. Свойство *ModelSpace* ... 655
8.2.369. Свойство *ModelType* ... 655

8.2.370. Свойство *ModelView* ... 656
8.2.371. Свойство *MomentOfInertia* ... 656
8.2.372. Свойство *MRUNumber* ... 656
8.2.373. Свойство *MSpace* ... 657
8.2.374. Свойство *MTextObjectId* ... 657
8.2.375. Свойство *MVertexCount* ... 657
8.2.376. Свойство *Name* ... 658
8.2.377. Свойство *NameNoMnemonic* ... 658
8.2.378. Свойство *NClose* ... 658
8.2.379. Свойство *NDensity* ... 659
8.2.380. Свойство *Normal* ... 659
8.2.381. Свойство *NumberOfControlPoints* ... 660
8.2.382. Свойство *NumberOfCopies* ... 660
8.2.383. Свойство *NumberOfFaces* ... 660
8.2.384. Свойство *NumberOfFitPoints* ... 660
8.2.385. Свойство *NumberOfLoops* ... 661
8.2.386. Свойство *NumberOfVertices* ... 661
8.2.387. Свойство *NumericScale* ... 661
8.2.388. Свойство *NVertexCount* ... 662
8.2.389. Свойство *ObjectID* ... 662
8.2.390. Свойство *ObjectName* ... 662
8.2.391. Свойство *ObjectSnapMode* ... 662
8.2.392. Свойство *ObjectSortByPlotting* ... 663
8.2.393. Свойство *ObjectSortByPSOutput* ... 663
8.2.394. Свойство *ObjectSortByRedraws* ... 663
8.2.395. Свойство *ObjectSortByRegens* ... 664
8.2.396. Свойство *ObjectSortBySelection* ... 664
8.2.397. Свойство *ObjectSortBySnap* ... 664
8.2.398. Свойство *ObliqueAngle* ... 665
8.2.399. Свойство *OleItemType* ... 665
8.2.400. Свойство *OLELaunch* ... 666
8.2.401. Свойство *OlePlotQuality* ... 666
8.2.402. Свойство *OLEQuality* ... 666
8.2.403. Свойство *OleSourceApp* ... 667
8.2.404. Свойство *OnMenuBar* ... 667
8.2.405. Свойство *OpenSave* ... 668
8.2.406. Свойство *Origin* ... 668
8.2.407. Свойство *OrthoOn* ... 668
8.2.408. Свойство *Output* ... 668
8.2.409. Свойство *OverrideCenter* ... 669
8.2.410. Свойство *OwnerID* ... 669
8.2.411. Свойство *PageSetupOverridesTemplateFile* ... 669
8.2.412. Свойство *PaperSpace* ... 670
8.2.413. Свойство *PaperUnits* ... 670
8.2.414. Свойство *Parent* ... 670

8.2.415. Свойство *Password* ... 671
8.2.416. Свойство *Path* ... 671
8.2.417. Свойство *PatternAngle* ... 671
8.2.418. Свойство *PatternDouble* ... 671
8.2.419. Свойство *PatternName* ... 672
8.2.420. Свойство *PatternScale* ... 672
8.2.421. Свойство *PatternSpace* ... 673
8.2.422. Свойство *PatternType* ... 673
8.2.423. Свойство *Perimeter* ... 673
8.2.424. Свойство *PickAdd* ... 674
8.2.425. Свойство *PickAuto* ... 674
8.2.426. Свойство *PickBoxSize* ... 674
8.2.427. Свойство *PickDrag* ... 675
8.2.428. Свойство *PickFirst* ... 675
8.2.429. Свойство *PickfirstSelectionSet* ... 675
8.2.430. Свойство *PickGroup* ... 675
8.2.431. Свойство *Plot* ... 676
8.2.432. Свойство *PlotConfigurations* ... 676
8.2.433. Свойство *PlotHidden* ... 676
8.2.434. Свойство *PlotLegacy* ... 677
8.2.435. Свойство *PlotLogFilePath* ... 677
8.2.436. Свойство *PlotOrigin* ... 677
8.2.437. Свойство *PlotPolicy* ... 678
8.2.438. Свойство *PlotRotation* ... 678
8.2.439. Свойство *PlotStyleName* ... 678
8.2.440. Свойство *Plottable* ... 679
8.2.441. Свойство *PlotType* ... 679
8.2.442. Свойство *PlotViewportBorders* ... 679
8.2.443. Свойство *PlotViewportsFirst* ... 680
8.2.444. Свойство *PlotWithLineweights* ... 680
8.2.445. Свойство *PlotWithPlotStyles* ... 680
8.2.446. Свойство *PolarTrackingVector* ... 681
8.2.447. Свойство *PostScriptPrologFile* ... 681
8.2.448. Свойство *Precision* ... 682
8.2.449. Свойство *Preferences* ... 682
8.2.450. Свойство *Preset* ... 682
8.2.451. Свойство *PrimaryUnitsPrecision* ... 683
8.2.452. Свойство *PrincipalDirections* ... 683
8.2.453. Свойство *PrincipalMoments* ... 684
8.2.454. Свойство *PrinterConfigPath* ... 684
8.2.455. Свойство *PrinterDescPath* ... 684
8.2.456. Свойство *PrinterPaperSizeAlert* ... 684
8.2.457. Свойство *PrinterSpoolAlert* ... 685
8.2.458. Свойство *PrinterStyleSheetPath* ... 685
8.2.459. Свойство *PrintFile* ... 686

8.2.460. Свойство *PrintSpoolerPath* ... 686
8.2.461. Свойство *PrintSpoolExecutable* ... 686
8.2.462. Свойство *ProductOfInertia* ... 686
8.2.463. Свойство *Profiles* ... 687
8.2.464. Свойство *PromptString* ... 687
8.2.465. Свойство *PropertyName* ... 687
8.2.466. Свойство *ProviderName* ... 688
8.2.467. Свойство *ProviderType* ... 688
8.2.468. Свойство *ProxyImage* ... 688
8.2.469. Свойство *QNewTemplateFile* ... 689
8.2.470. Свойство *QuietErrorMode* ... 689
8.2.471. Свойство *RadiiOfGyration* ... 689
8.2.472. Свойство *Radius* ... 690
8.2.473. Свойство *RadiusRatio* ... 690
8.2.474. Свойство *ReadOnly* ... 690
8.2.475. Свойство *Red* ... 690
8.2.476. Свойство *ReferenceCount* ... 691
8.2.477. Свойство *RegenerateTableSuppressed* ... 691
8.2.478. Свойство *RegisteredApplications* ... 691
8.2.479. Свойство *RenderSmoothness* ... 691
8.2.480. Свойство *RevisionNumber* ... 692
8.2.481. Свойство *Rotation* ... 692
8.2.482. Свойство *RoundDistance* ... 692
8.2.483. Свойство *RowHeight* ... 693
8.2.484. Свойство *Rows* ... 693
8.2.485. Свойство *RowSpacing* ... 693
8.2.486. Свойство *SaveAsType* ... 693
8.2.487. Свойство *Saved* ... 694
8.2.488. Свойство *SavePreviewThumbnail* ... 695
8.2.489. Свойство *ScaleFactor* ... 695
8.2.490. Свойство *ScaleHeight* ... 695
8.2.491. Свойство *ScaleLineweights* ... 696
8.2.492. Свойство *ScaleWidth* ... 696
8.2.493. Свойство *Schema* ... 696
8.2.494. Свойство *SCMCommandMode* ... 697
8.2.495. Свойство *SCMDefaultMode* ... 697
8.2.496. Свойство *SCMEditMode* ... 697
8.2.497. Свойство *SCMTimeMode* ... 698
8.2.498. Свойство *SCMTimeValue* ... 698
8.2.499. Свойство *SecondPoint* ... 698
8.2.500. Свойство *SegmentPerPolyline* ... 699
8.2.501. Свойство *Selection* ... 699
8.2.502. Свойство *SelectionSets* ... 699
8.2.503. Свойство *SerialNumber* ... 699
8.2.504. Свойство *ShadePlot* ... 700

8.2.505. Свойство *SheetView* ... 700
8.2.506. Свойство *ShortcutMenu* ... 700
8.2.507. Свойство *ShortCutMenuDisplay* ... 701
8.2.508. Свойство *Show* ... 701
8.2.509. Свойство *ShowPlotStyles* ... 701
8.2.510. Свойство *ShowProxyDialogBox* ... 702
8.2.511. Свойство *ShowRasterImage* ... 702
8.2.512. Свойство *ShowRotation* ... 702
8.2.513. Свойство *ShowWarningMessages* ... 703
8.2.514. Свойство *SingleDocumentMode* ... 703
8.2.515. Свойство *SnapBasePoint* ... 704
8.2.516. Свойство *SnapOn* ... 704
8.2.517. Свойство *SnapRotationAngle* ... 704
8.2.518. Свойство *SolidFill* ... 705
8.2.519. Свойство *StandardScale* ... 705
8.2.520. Свойство *StartAngle* ... 706
8.2.521. Свойство *StartParameter* ... 707
8.2.522. Свойство *StartPoint* ... 707
8.2.523. Свойство *StartTangent* ... 707
8.2.524. Свойство *StatusId* ... 707
8.2.525. Свойство *StoreSQLIndex* ... 708
8.2.526. Свойство *StyleName* ... 708
8.2.527. Свойство *StyleSheet* ... 709
8.2.528. Свойство *Subject* ... 709
8.2.529. Свойство *SubMenu* ... 709
8.2.530. Свойство *SummaryInfo* ... 709
8.2.531. Свойство *SupportPath* ... 710
8.2.532. Свойство *SuppressLeadingZeros* ... 710
8.2.533. Свойство *SuppressTrailingZeros* ... 710
8.2.534. Свойство *SuppressZeroFeet* ... 711
8.2.535. Свойство *SuppressZeroInches* ... 711
8.2.536. Свойство *SymbolPosition* ... 711
8.2.537. Свойство *System* ... 712
8.2.538. Свойство *Table* ... 712
8.2.539. Свойство *TablesReadOnly* ... 712
8.2.540. Свойство *TableStyleOverrides* ... 713
8.2.541. Свойство *TabOrder* ... 713
8.2.542. Свойство *TagString* ... 713
8.2.543. Свойство *Target* ... 714
8.2.544. Свойство *TempFileExtension* ... 714
8.2.545. Свойство *TempFilePath* ... 714
8.2.546. Свойство *TemplateDwgPath* ... 715
8.2.547. Свойство *TempXrefPath* ... 715
8.2.548. Свойство *Text* ... 715
8.2.549. Свойство *TextAlignmentPoint* ... 716

8.2.550. Свойство *TextAttachment* ... 716
8.2.551. Свойство *TextColor* ... 716
8.2.552. Свойство *TextEditor* ... 717
8.2.553. Свойство *TextFont* ... 717
8.2.554. Свойство *TextFontSize* ... 717
8.2.555. Свойство *TextFontStyle* ... 717
8.2.556. Свойство *TextFrameDisplay* ... 718
8.2.557. Свойство *TextGap* ... 718
8.2.558. Свойство *TextGenerationFlag* ... 718
8.2.559. Свойство *TextHeight* ... 719
8.2.560. Свойство *TextInside* ... 719
8.2.561. Свойство *TextInsideAlign* ... 720
8.2.562. Свойство *TextMovement* ... 720
8.2.563. Свойство *TextOutsideAlign* ... 721
8.2.564. Свойство *TextOverride* ... 721
8.2.565. Свойство *TextPosition* ... 721
8.2.566. Свойство *TextPrecision* ... 722
8.2.567. Свойство *TextPrefix* ... 722
8.2.568. Свойство *TextRotation* ... 722
8.2.569. Свойство *TextString* ... 723
8.2.570. Свойство *TextStyle* ... 723
8.2.571. Свойство *TextStyles* ... 723
8.2.572. Свойство *TextSuffix* ... 723
8.2.573. Свойство *TextureMapPath* ... 724
8.2.574. Свойство *TextWinBackgrndColor* ... 724
8.2.575. Свойство *TextWinTextColor* ... 724
8.2.576. Свойство *Thickness* ... 725
8.2.577. Свойство *TimeServer* ... 725
8.2.578. Свойство *TimeStamp* ... 725
8.2.579. Свойство *TipOffset* ... 726
8.2.580. Свойство *Title* ... 726
8.2.581. Свойство *TitleSuppressed* ... 726
8.2.582. Свойство *ToleranceDisplay* ... 727
8.2.583. Свойство *ToleranceHeightScale* ... 727
8.2.584. Свойство *ToleranceJustification* ... 728
8.2.585. Свойство *ToleranceLowerLimit* ... 728
8.2.586. Свойство *TolerancePrecision* ... 728
8.2.587. Свойство *ToleranceSuppressLeadingZeros* ... 729
8.2.588. Свойство *ToleranceSuppressTrailingZeros* ... 729
8.2.589. Свойство *ToleranceSuppressZeroFeet* ... 729
8.2.590. Свойство *ToleranceSuppressZeroInches* ... 730
8.2.591. Свойство *ToleranceUpperLimit* ... 730
8.2.592. Свойство *Toolbars* ... 730
8.2.593. Свойство *ToolPalettePath* ... 731
8.2.594. Свойство *Top* ... 731

8.2.595. Свойство *TotalAngle* ... 731
8.2.596. Свойство *TranslateIDs* ... 732
8.2.597. Свойство *Transparency* ... 732
8.2.598. Свойство *TrueColor* ... 732
8.2.599. Свойство *TrueColorImages* ... 733
8.2.600. Свойство *TwistAngle* ... 733
8.2.601. Свойство *Type* ... 733
8.2.602. Свойство *UCSIconAtOrigin* ... 735
8.2.603. Свойство *UCSIconOn* ... 736
8.2.604. Свойство *UCSPerViewport* ... 736
8.2.605. Свойство *Units* ... 736
8.2.606. Свойство *UnitsFormat* ... 736
8.2.607. Свойство *UnitsType* ... 737
8.2.608. Свойство *Updatable* ... 737
8.2.609. Свойство *UpperRightCorner* ... 738
8.2.610. Свойство *UpsideDown* ... 738
8.2.611. Свойство *URL* ... 738
8.2.612. Свойство *URLDescription* ... 739
8.2.613. Свойство *URLNamedLocation* ... 739
8.2.614. Свойство *Used* ... 739
8.2.615. Свойство *UseLastPlotSettings* ... 739
8.2.616. Свойство *User* ... 740
8.2.617. Свойство *UserCoordinateSystems* ... 740
8.2.618. Свойство *UseStandardScale* ... 740
8.2.619. Свойство *Utility* ... 741
8.2.620. Свойство *Value* ... 741
8.2.621. Свойство *VBE* ... 741
8.2.622. Свойство *Verify* ... 742
8.2.623. Свойство *Version* ... 742
8.2.624. Свойство *VersionGuid* ... 742
8.2.625. Свойство *VertCellMargin* ... 742
8.2.626. Свойство *VerticalTextPosition* ... 743
8.2.627. Свойство *ViewportDefault* ... 743
8.2.628. Свойство *ViewportOn* ... 744
8.2.629. Свойство *Viewports* ... 744
8.2.630. Свойство *Views* ... 744
8.2.631. Свойство *ViewToPlot* ... 744
8.2.632. Свойство *VisibilityEdge1* ... 745
8.2.633. Свойство *VisibilityEdge2* ... 745
8.2.634. Свойство *VisibilityEdge3* ... 745
8.2.635. Свойство *VisibilityEdge4* ... 745
8.2.636. Свойство *Visible* ... 746
8.2.637. Свойство *Volume* ... 746
8.2.638. Свойство *Weights* ... 746
8.2.639. Свойство *Width* ... 747

8.2.640. Свойство *WindowLeft* ... 747
8.2.641. Свойство *WindowState* ... 747
8.2.642. Свойство *WindowTitle* ... 748
8.2.643. Свойство *WindowTop* ... 748
8.2.644. Свойство *WorkspacePath* ... 748
8.2.645. Свойство *XEffectiveScaleFactor* ... 748
8.2.646. Свойство *XRefDatabase* ... 749
8.2.647. Свойство *XrefDemandLoad* ... 749
8.2.648. Свойство *XRefEdit* ... 749
8.2.649. Свойство *XRefFadeIntensity* ... 750
8.2.650. Свойство *XRefLayerVisibility* ... 750
8.2.651. Свойство *XScaleFactor* ... 750
8.2.652. Свойство *XVector* ... 750
8.2.653. Свойство *YEffectiveScaleFactor* ... 751
8.2.654. Свойство *YScaleFactor* ... 751
8.2.655. Свойство *YVector* ... 751
8.2.656. Свойство *ZEffectiveScaleFactor* ... 752
8.2.657. Свойство *ZScaleFactor* ... 752

Глава 9. События и реакторы ... 753

9.1. Обрабатываемые события ... 753
9.2. Классификация реакторов ... 756
9.3. Справочные функции реакторов ... 758
9.3.1. Функция *vlr-types* ... 758
9.3.2. Функция *vlr-reactors* ... 758
9.3.3. Функция *vlr-type* ... 759
9.4. Действия реакторов ... 759
9.4.1. Функция *vlr-reaction-names* ... 759
9.4.2. Функция *vlr-beep-reaction* ... 760
9.4.3. Функция *vlr-trace-reaction* ... 760
9.5. Создание реакторов ... 760
9.5.1. Функция *vlr-acdb-reactor* ... 761
9.5.2. Функция *vlr-docmanager-reactor* ... 762
9.5.3. Функция *vlr-linker-reactor* ... 763
9.5.4. Функция *vlr-object-reactor* ... 764
9.5.5. Функция *vlr-editor-reactor* ... 765
9.5.6. Функция *vlr-command-reactor* ... 768
9.5.7. Функция *vlr-deepclone-reactor* ... 768
9.5.8. Функция *vlr-dwg-reactor* ... 769
9.5.9. Функция *vlr-dxf-reactor* ... 770
9.5.10. Функция *vlr-insert-reactor* ... 771
9.5.11. Функция *vlr-lisp-reactor* ... 772
9.5.12. Функция *vlr-miscellaneous-reactor* ... 773
9.5.13. Функция *vlr-mouse-reactor* ... 774
9.5.14. Функция *vlr-sysvar-reactor* ... 774

9.5.15. Функция *vlr-toolbar-reactor* ... 775
9.5.16. Функция *vlr-undo-reactor* ... 776
9.5.17. Функция *vlr-wblock-reactor* ... 777
9.5.18. Функция *vlr-window-reactor* ... 779
9.5.19. Функция *vlr-xref-reactor* ... 780
9.5.20. Функция *vlr-reaction-set* ... 783
9.6. Операции с реакторами ... 784
9.6.1. Функция *vlr-reactions* ... 784
9.6.2. Функция *vlr-remove* ... 784
9.6.3. Функция *vlr-remove-all* ... 784
9.6.4. Функция *vlr-add* ... 785
9.6.5. Функция *vlr-added-p* ... 785
9.6.6. Функция *vlr-current-reaction-name* ... 785
9.6.7. Функция *vlr-data* ... 785
9.6.8. Функция *vlr-data-set* ... 785
9.6.9. Функция *vlr-owner-add* ... 786
9.6.10. Функция *vlr-owner-remove* ... 786
9.6.11. Функция *vlr-owners* ... 786
9.6.12. Функция *vlr-notification* ... 786
9.6.13. Функция *vlr-set-notification* ... 786
9.7. Временные и постоянные реакторы ... 787
9.7.1. Функция *vlr-pers* ... 787
9.7.2. Функция *vlr-pers-release* ... 787
9.7.3. Функция *vlr-pers-p* ... 788
9.7.4. Функция *vlr-pers-list* ... 788
9.8. Правила работы с реакторами ... 788
9.9. Примеры реакторов ... 789
9.9.1. Схема создания реакторов ... 789
9.9.2. Реактор изменений базы рисунка ... 790
9.9.3. Реактор DXF-операций ... 791
9.9.4. Реактор команды ERASE ... 792
9.9.5. Отслеживание двойного щелчка (реактор мыши) ... 793
9.9.6. Реактор отрезков ... 794
9.9.7. Связанные окружности (реактор объектов) ... 797

Глава 10. Приемы программирования ... 800

10.1. Рекурсия ... 800
10.1.1. Функция *PL:String-Rep* ... 801
10.1.2. Функция *PL:String->List* ... 802
10.1.3. Функция *PL:ReadFile* ... 803
10.1.4. Набор функций для работы с регулярными списками ... 804
10.2. Выбор примитивов методами ActiveX ... 810
10.2.1. Интерактивный выбор ... 812
10.2.2. Программный выбор ... 812

10.3. Особенности обработки семейств объектов 813
10.4. Пример работы с внешним приложением (чтение файла Excel 2003) 815
10.4.1. Анализ методов и свойств приложения Excel 816
10.4.2. Тестовый файл книги Excel 816
10.4.3. Тексты программ 817
10.4.4. Анализ программ 823

Послесловие 825

Приложение 1. Алфавитный перечень функций 826

Приложение 2. DXF-коды объектов AutoCAD 844

Общие DXF-коды 844
Символьные таблицы 852
Некоторые примитивы 854

Приложение 3. Индикатор 861

Приложение 4. Светофор 864

Приложение 5. Танцы панелей 870

Приложение 6. Пересечение объектов 878

Приложение 7. Очистка расширенных данных 881

Приложение 8. Метод *CopyObjects* 884

Приложение 9. Описание компакт-диска 890

Перечень материалов 890
Использование материалов 892

Литература 893

Предметный указатель 894

Предисловие

Система AutoCAD в настоящее время очень широко используется во всем мире для создания графических документов самых различных отраслей: машиностроения, архитектуры, строительства, связи и т. д. Этому способствует удивительная гибкость системы, многочисленные возможности настройки и адаптации, а также инструменты разработки пользовательских приложений. Рассматриваемый в данной книге язык AutoLISP, являющийся вариантом стандартного языка LISP и появившийся в системе AutoCAD версии 2.1 еще в 1982 году, всегда был основным языком создания дополнительных возможностей пользователя. Он достаточно прост в изучении и освоении, но в то же время предоставляет доступ ко всем основным элементам и объектам системы. Спустя десять лет фирма Autodesk сначала расширила язык за счет средств Visual LISP, заимствованных из языка Vital LISP, а затем добавила средства ActiveX Automation, предоставляющие доступ к объектной модели AutoCAD.

Эта книга предназначена для тех, кто уже имеет опыт работы в системе AutoCAD и собирается создавать свои приложения именно на языке LISP (неважно, будут это средства AutoLISP или Visual LISP). Другие средства (VBA, ObjectARX, .NET Framework и т. д.) тоже достойны внимания, но в данной книге не рассматриваются. О них можно прочитать, например, в книгах Н. Н. Полещука "AutoCAD 2004: разработка приложений и адаптация"[1] и "AutoCAD: разработка приложений, настройка и адаптация"[2].

Изложение в настоящей книге ведется на базе двух версий: 2005 и 2006. Они к моменту написания книги были последними версиями системы AutoCAD. Однако материалы книги будут полезны и тем, кто работает с другими версиями.

---

[1] Полещук Н. Н. AutoCAD 2004: разработка приложений и адаптация. — СПб.: БХВ-Петербург, 2004. — 624 с.

[2] Полещук Н. Н. AutoCAD: разработка приложений, настройка и адаптация. — СПб.: БХВ-Петербург, 2006. — 992 с. (+CD-ROM).

Для того чтобы читателю легче было увидеть отличия системы AutoCAD 2005 и 2006 от AutoCAD 2004, все наиболее интересные изменения помечаются соответствующими значками: 2005 или 2006.

Данная книга может служить справочным документом для тех, кто уже работает в среде AutoCAD с языком LISP (AutoLISP или Visual LISP), но также и справочно-методическим пособием для тех, кто только приступает к изучению языка. Этому помогут разделы "Интерпретатор LISP", "Разработка функций пользователя", "Функции доступа к примитивам AutoCAD" и другие аналогичные разделы, с изложением теоретических основ (в том числе структуры объектов рисунка). Кроме того, в книге приведено и подробно описано большое количество программных листингов, тексты которых можно найти на сопроводительном компакт-диске.

Книга разделена на десять глав. В *главе 1* излагаются принципы работы интерпретатора LISP-программ и описываются базовые функции языка AutoLISP. *Глава 2* посвящена среде разработки приложений, предоставляющей сервисные возможности написания, проверки и отладки программ, а также компиляции FAS- и VLX-приложений. *Глава 3* связана с диалоговыми окнами на языке DCL, которые можно использовать в приложениях.

Глава 4 посвящена вопросам взаимодействия с меню и адаптациями. Проблемы построения справочных систем даются в *главе 5*.

Одним из главных преимуществ Visual LISP является возможность применять технологию ActiveX, работать с объектной моделью AutoCAD. Этому посвящены *глава 6*, а также *главы 7, 8* с энциклопедическим изложением методов и свойств объектов и семейств AutoCAD. Каждое свойство и каждый метод сопровождаются поясняющим примером.

В *главе 9* приводится материал о специальных программных объектах (реакторах) и событиях как элементах объектной модели AutoCAD. В *главе 10* показаны малоизвестные приемы программирования (рекурсивные функции, применение наборов в ActiveX), а также техника взаимодействия LISP-приложения с внешним приложением (на примере Microsoft Excel 2003).

Справочные *приложения 1 и 2* содержат перечень функций и таблицы DXF-кодов основных объектов AutoCAD. Остальные приложения отданы полезным примерам.

К книге прилагается компакт-диск, на котором приведены тексты функций и описания диалоговых окон, разобранные в листингах.

В книге используются следующие приемы оформления:

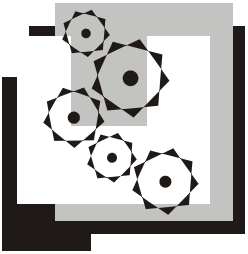
- имена переменных, объектов, методов, свойств, функций приведены шрифтом Courier, например: `g_name`, `Documents`, `ClipBoundary`, `Length`;
- имена английских команд даются в верхнем регистре, а соответствующих русских команд — в верхнем регистре и в скобках, например: PEDIT (ПОЛРЕД);

- имена системных переменных приводятся в верхнем регистре, например: DIMSCALE;
- типы примитивов представлены в верхнем регистре, например: ACAD_TABLE;
- названия диалоговых окон, опций команд выделены полужирным шрифтом, их русские аналоги — обычным шрифтом, например: **Fit** (СГладить);
- определяемые термины и ссылки на другие главы и разделы выделены курсивом, например: *реактор, глава 2*.

Главы 7 и 8, а также разд. 10.1—10.3 написаны П. В. Лоскутовым, остальные материалы — Н. Н. Полещуком. Материал приложения 8 предоставил Э. В. Смолянка (Санкт-Петербург), которому авторы выражают свою благодарность.

Дополнительные сведения о других книгах авторов можно найти на сайтах **www.private.peterlink.ru/poleshchuk/cad**, **www.cad.dp.ua/poleschuk.html**, **poleshchuk.spb.ru** и **www.kurganobl.ru/cad** (на русском языке), **www.private.peterlink.ru/poleshchuk/cad/eng**, **www.cad.dp.ua/english/poleschuk_e.html** и **partnerproducts.autodesk.com/popups/company.asp?rdid=DERU1314** (на английском языке).

ГЛАВА 1



Языки AutoLISP и Visual LISP

Язык программирования AutoLISP — это вариант языка LISP, в который добавлены функции доступа к объектам, таблицам и словарям системы AutoCAD. Основой языка LISP является работа со списками, которые могут иметь произвольную длину и включать элементы разной природы (числа, текстовые строки, указатели файлов и т. д.). В то же время в языке доступны обычные арифметические вычисления, логические операции, работа с файлами и т. д. С помощью языка AutoLISP можно писать программы или вводить в командной строке выражения, которые затем вычисляет система AutoCAD. Для чтения и разбора выражений в систему AutoCAD встроен интерпретатор AutoLISP.

За последние годы язык развился за счет новых функций с префиксом `vl-`, а также функций, предоставляющих доступ к объектной модели AutoCAD и реализующих элементы технологии ActiveX. Такие функции (довольно условно) отнесены к языку Visual LISP. Для простоты объединение языков AutoLISP и Visual LISP будем называть просто LISP.

В данной главе изучаются основные функции языка LISP и приводятся практические примеры. В *главах 3* и *4* будет рассмотрено применение LISP в файлах меню и программирование диалоговых окон с помощью LISP и языка DCL.

Для разработки и отладки программ на LISP удобно пользоваться входящей в систему AutoCAD интегрированной средой разработки приложений, изучаемой в *главе 2*.

Материал излагается по состоянию версии AutoCAD 2006. Изменения, связанные с версиями 2005 и 2006, как правило, в тексте подчеркнуты особо.

1.1. Интерпретатор LISP

В AutoCAD входит интерпретатор LISP, начинающий работать, если вы в командной строке в ответ на стандартную подсказку **Command:** *(Команда:)* введе-

те открывающую скобку — "(". В этот момент AutoCAD настраивается на *ввод* и *вычисление LISP-выражения*, которое должно иметь форму списка, т. е. начинаться открывающей скобкой и заканчиваться закрывающей, а элементы списка должны отделяться друг от друга пробелами. Например, для получения суммы трех чисел вы можете ввести:

```
(+  62.8  37.2  1923.5)
```

Как только вы завершите ввод своего выражения нажатием клавиши <Enter> или пробелом, AutoCAD вычислит указанное выражение и выдаст результат (рис. 1.1): `2023.5` (напомним, что десятичная точка в вещественных числах играет роль разделителя между целой и дробной частями).

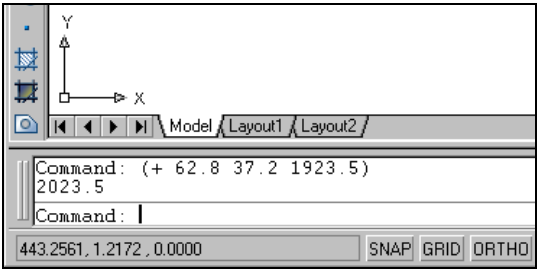


Рис. 1.1. Использование вычислителя выражений интерпретатора LISP

Замечания

- Клавиатурный ввод выражений LISP встречается сравнительно редко. Обычно выражения LISP записываются в текстовый файл с расширением lsp, который затем загружается в AutoCAD с помощью функции `load` *(см. разд. 1.2.4)*.
- При работе с длинными числами система AutoCAD отображает в командной строке усеченный результат (обычно не более шести значащих цифр). При этом большие числа показываются в научной форме, например: 7.7791e+007. Однако если результат вычисления выражения сохраняется в переменной *(см. далее)*, то он имеет максимально возможную точность.

1.1.1. Вычисляемые выражения

В приведенном примере для вычисления было предложено выражение, открывающееся и закрывающееся круглыми скобками и состоящее из четырех элементов: первый — имя функции сложения (+), остальные — аргументы функции, т. е. слагаемые (62.8, 37.2 и 1923.5). Это иллюстрирует форму вычисляемых выражений языка AutoLISP:

```
(<функция> [<аргумент1> [<аргумент2> ... [<аргументN>]...]])
```

Здесь `<функция>` — это имя функции, `<аргумент1>`, `<аргумент2>`, ..., `<аргументN>` — аргументы функции, разделяемые, по крайней мере, одним про-

белом. Квадратные скобки в приведенной форме указывают на возможное отсутствие находящихся между ними аргументов. Количество аргументов зависит от синтаксиса функции. Существуют функции, которые вызываются без аргументов. Алфавитный список базовых функций LISP (без функций методов и свойств, рассматриваемых в *главах 7* и *8*) приведен в *приложении 1*.

С точки зрения LISP все, что заключено в круглых скобках, является списком, который следует обработать как вычисляемое выражение.

Основное правило AutoLISP — баланс скобок, т. е. количество открывающих скобок должно быть равно количеству закрывающих. Иначе AutoCAD, обнаружив дисбаланс (например, если пользователь забыл ввести конечную скобку), выдаст об этом сообщение (рис. 1.2).

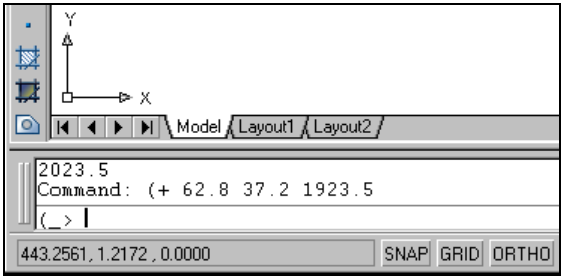


Рис. 1.2. Сообщение о дисбалансе скобок

Для разрешения ситуации, изображенной на рис. 1.2, необходимо ввести закрывающую скобку и нажать клавишу <Enter> или пробел. После этого вы получите результат вычисления. В интегрированной среде разработки Visual LISP *(см. главу 2)* предусмотрены механизмы проверки скобок в файлах с текстами программ.

В качестве любого аргумента функции может быть введено некоторое вычисляемое выражение, ограниченное скобками, или символ LISP (о символах *см. разд. 1.2*).

Выражения LISP могут быть как простыми, так и составными, т. е. в них могут в качестве элементов использоваться другие выражения, причем разделяющими знаками в выражениях являются круглые скобки, пробелы и двойные кавычки (двойные кавычки должны ограничивать с двух сторон текстовые константы — о текстовых или строковых константах *см. разд. 1.2*). Например:

```
(-  (+  14.2  3.09)  (*  12.5  0.93))
```

Здесь "-" — имя функции (вычитание), имеющей два аргумента в виде выражений, первое из которых является выражением с операцией сложения (в нем использована функция сложения "+"), а второе — выражением с операцией умножения (в нем использована функция умножения "*"). Сначала интерпретатор языка LISP вычислит выражения во внутренних скобках (в нашем примере полу-

чатся значения 17.29 и 11.625), а затем подставит эти значения и вычислит выражение `(- 17.29 11.625)`, что должно дать результат 5.665 (вещественное число).

1.1.2. Функция *setq* и точность отображения

Интерпретатор LISP вычисляет значение каждого введенного выражения и возвращает его в среду AutoCAD. *Возвращаемое значение*, в свою очередь, либо может быть передано для дальнейших вычислений в выражение более высокого уровня, как в последнем примере *разд. 1.1.1* (аргументы функции верхнего уровня были вычисляемыми выражениями), либо может быть сохранено в *переменной* (человеку, знакомому с программированием, ясно, что это — именованные участки памяти для хранения данных) с помощью функции присвоения `setq`, имеющей следующий синтаксис:

```
(setq <переменная1> <выражение1> [<переменная2> <выражение2> ...
[<переменнаяN> <выражениеN>] ... ])
```

Функция `setq` может использоваться с любым количеством аргументов, которое должно быть обязательно четным и не менее двух. В качестве аргументов `<переменная1>`, `<переменная2>` и т. д. могут использоваться любые символы (имена) LISP, не занятые в наименованиях функций. О символах речь пойдет в *разд. 1.2.* В качестве аргументов `<выражение1>`, `<выражение2>` и т. д. могут использоваться выражения LISP, имена переменных или константы (например, числа).

Функция `setq` — основное средство для сохранения значений, возвращаемых другими выражениями. Пример:

```
(setq myd (- (+ 6.832 200.1) (+ 423.1 33.22)))
```

Значение выражения `(- (+ 6.832 200.1) (+ 423.1 33.22))` сохранено в переменной `myd`.

Внимание!

Следует иметь в виду, что значение, отображенное в командной строке, может не совпасть с реальным значением выражения. Вызвано это тем, что по умолчанию в вещественных числах система AutoCAD **показывает в командной строке не более шести значащих цифр**. Например, если бы в итоге вычисления должен был бы получиться результат 329.65142, то все равно вы бы увидели в командной строке только 329.651. Однако внутри программы AutoCAD оперирует числами **с максимальной** точностью (15 знаков после десятичной точки).

Иллюстрация к сделанному замечанию. Попробуйте ввести в командной строке выражение

```
(setq w 329.65142)
```

Вы увидите, что AutoCAD в качестве результата покажет вам `329.651`. Но это совсем не означает, что система испортила значение переменной `w`.

Для проверки введите еще одно выражение:

```
(- w 329.0)
```

И получите результат с "потерянными" значащими цифрами: `0.65142`.

В дальнейшем можно прочитать значение вычисленной ранее переменной `w` (не забывая о точности отображения!) с помощью операции `!` (восклицательный знак) и имени переменной. Если в командной строке AutoCAD в ответ на стандартную подсказку **Command:** *(Команда:)* ввести `!w`, то AutoCAD выдаст текущее значение этой переменной (в нашем примере: `329.651`).

1.1.3. Комментарии

В текст программы полезно вставлять *комментарии*, поясняющие, например, особенности алгоритма. Комментарием считается часть строки, начиная с символа ";" (точка с запятой), после которого не следует сразу же символ "|" (вертикальная черта), и кончая последним символом строки. Закомментированной может быть вся строка программы.

Если комментарий нужно вставить в середину строки, тогда началом комментария должны быть символы ";|", а концом комментария — символы "|;" (причем конец комментария может располагаться на другой строке). Примеры (комментарии выделены полужирным шрифтом):

```
; Закомментирована вся строка
(setq two_pi (+ pi pi))        ; Закомментирован конец строки
(setq uv0 (+ 44.73 ;| внутренний
 комментарий |;206.7))
(command "_.CIRCLE" '(14.0 6.6) 150.55) ; рисование окружности
```

1.2. Символы

Символы LISP — это слова, состоящие из группы буквенно-цифровых знаков и являющиеся именами функций или переменных, используемых для хранения данных пользователя. В символах LISP не должны использоваться разделители выражений, к которым относятся круглые скобки, пробелы и двойные кавычки, и служебные знаки — "." (точка), "," (запятая), ";" (точка с запятой), "'" (апостроф), "/" (косая черта), "\" (обратная косая черта). Написание символов не должно соответствовать написанию чисел (например, символ не должен состоять из одних цифр).

Зарезервированными являются символы, используемые как имена функций, а также некоторые обозначения констант *(см. разд. 1.2.2)*. Алфавитный перечень базовых функций языка AutoLISP приведен в *приложении 1.*

Замечание

В именах символов LISP регистр (верхний или нижний) не имеет значения. Поэтому `setq`, `SETQ`, `SETq`, `Setq`, `sETQ` обозначают одну и ту же функцию `setq`.

1.2.1. Типы данных

LISP работает с данными следующих типов:

- целое число;
- вещественное число;
- строка;
- список, точечная пара, точечный список;
- дескриптор файла;
- указатель функции, приложения;
- примитив AutoCAD;
- набор.

Кроме того, в главах, посвященных технологии ActiveX, описываются еще некоторые типы данных (варианты, безопасные массивы, VLA-объекты, VLR-объекты).

Замечание

Тип символа может быть получен с помощью функции `type` *(см. разд. 1.7.1)*.

Символам LISP могут быть присвоены значения любых типов (этим язык LISP выгодно отличается от языков высокого уровня типа Fortran). Символ языка LISP может в ходе программы менять тип своего значения. Пользователь в явном виде может задавать константы следующих типов: целые, вещественные, строковые и списковые.

Целые числа могут быть как положительными, так и отрицательными или нулями. Диапазон допустимых значений — от –2 147 483 648 до +2 147 483 647.

Замечание

Функция `getint`, запрашивающая ввод целого числа, позволяет использовать значения только в диапазоне между –32 768 и +32 767. Аналогичный диапазон значений целых чисел допускался в языке LISP ранних версий.

К целым числам относятся слова, состоящие лишь из цифр. Кроме того, целые числа могут иметь знак (+ или –). Если пользователь пытается задать целое число, не лежащее между описанными выше предельными значениями, то вводимое значение преобразуется в вещественное.

Вещественные числа — это слова, которые состоят из цифр и содержат внутри только одну десятичную точку. Могут иметь знак (+ или –). В состав веществен-

ного числа может входить латинский символ `e` или `E` с показателем степени числа 10 (научная форма представления вещественных чисел).

Вещественные числа не должны начинаться с десятичной точки (например, вместо `.5` следует писать `0.5`). Допустимые значения вещественных чисел находятся в диапазоне от –1.7976931348623158e+308 до +1.7976931348623158e+308.

Строковые константы начинаются и заканчиваются кавычками (сами кавычки в длину константы не входят) и содержат внутри себя текст, который может состоять из любых знаков, в том числе пробелов, скобок, апострофов. Особо рассматривается знак "\", который является служебным и предназначен для ввода в текстовую строку некоторых специальных знаков (например, кавычек и самого знака "\"). Константа "" считается строкой нулевой длины. В строковые константы могут входить управляющие знаки печати, например: "\n" — переход на новую строку, "\t" — символ табуляции.

Списковые константы — это списки, состоящие из постоянных элементов любого типа (списки, как и функции, открываются и закрываются круглыми скобками). Для того чтобы AutoCAD не вычислял выражение как функцию, постоянные списки предваряются апострофом перед открывающей скобкой.

Примеры констант:

- `91`
- `-25408`
- `539.3582`
- `777.5e+3`
- `"Версия для печати"`
- `"336\\_5"` (первая косая черта является служебной, вторая — значащим символом; если распечатать такую строку, то получим `336\_5`)
- `'(0.0 0.0 1.0)`
- `'("Аа" "Бб" "Вв" "Гг" "Дд" "Ее")`
- `'(0 75 "chair" 25 "window" 3 "" "&")`

Особые списки, в которых только два элемента и разделителем между ними является точка, называются *точечными парами*. Они создаются с помощью функции `cons`. Примеры точечных пар:

- `(0 . "POLYLINE")`
- `(62 . 29)`

Списки, которые состоят из трех или более элементов и у которых последний элемент отделяется точкой, называются *точечными списками*. Они могут создаваться с помощью функции `vl-list*`. Пример:

- `(120 14 . 67)`

Функция setq возвращает значение, совпадающее со значением выражения, присваиваемого последней переменной (<переменнаяN>), поэтому тип результата совпадает с типом этого выражения.

Есть функции, которые преобразуют значение из одного типа в другой (например, функции atoi, fix, float, rtos и др.). Эти функции рассматриваются в *разд. 1.7.*

1.2.2. Предопределенные символы

К основным зарезервированным символам, значения которых пользователям не рекомендуется изменять, помимо имен функций, относятся следующие:

- T — единственная логическая константа, имеющая значение "истина";
- nil — пустое значение (оно же в логических операциях играет роль и значения "ложь"); это значение не эквивалентно числу 0 или пустой строке ("");
- pi — постоянная, равная отношению длины окружности к диаметру (3.14159...);
- pause — постоянная, позволяющая ввести паузу при работе команды AutoCAD (используется в функции command).

Представленные четыре символа называются еще *предопределенными символами.*

Замечание

Функциям, требующим в качестве аргумента логическое значение "истина", вместо T можно передавать любое значение, отличное от nil (например, число, строку и т. д.).

1.2.3. Переменные

Для хранения данных пользователь может вводить свои символы (переменные), не совпадающие по написанию с зарезервированными или ранее занятыми. Создание новых переменных осуществляется с помощью функции setq, например:

```
(setq r1 15.33 s24 9)
```

Здесь вводятся переменные r1 и s24, получающие, соответственно, значения 15.33 (вещественное) и 9 (целое). При этом к той части оперативной памяти, которая отведена для текущего рисунка, добавляются участки, занимаемые переменными. Если переменной присвоить значение nil, то такая переменная из памяти удаляется, и ее место освобождается для других операций LISP.

Замечание

В многодокументном режиме (когда в одном сеансе AutoCAD открыто несколько рисунков) переменные одного рисунка не доступны в другом рисунке. Для передачи

требуется экспортировать переменные во внедокументное пространство имен *(см. разд. 1.17.33)*.

Переменные могут использоваться в любых выражениях, например:

```
(setq r61 (+ j 81.4 ysa))
```

1.2.4. Функция *load*

Ввод с клавиатуры выражений LISP не всегда удобен, поэтому можно записать предварительно свои строки на языке LISP в файл с расширением lsp (текст располагается в файле произвольным образом построчно, не обязательно каждое новое выражение начинать с новой строки; желательно заканчивать строку на разделителе — пробеле или скобке), а затем загрузить этот файл с помощью функции `load`. Синтаксис функции следующий:

```
(load <файл> [<ошибка>])
```

Здесь `<файл>` — это строка с именем загружаемого файла, а `<ошибка>` — это выражение, которое нужно выполнить, или значение (например, строка с текстом сообщения), которое нужно вывести, если загружаемый файл не будет обнаружен. Возвращаемое значение в случае удачной загрузки — результат последнего выражения или имя последней функции, если в файле находятся определения функций (см. функцию `defun`). В случае если загружаемый файл не найден, то выполняется `<ошибка>` (оно может быть текстом сообщения об ошибке).

Например:

```
(load "e25run.lsp" "Файл не найден")
```

Аргумент `<ошибка>` может быть опущен, в этом случае при ошибке загрузки выдается стандартное сообщение AutoCAD. Аргумент `<файл>` обычно содержит имя файла, который ищется в соответствии с настройкой рабочего каталога и путей поиска файлов поддержки. Однако можно указывать и полное имя с путем — например, `"c:/D_Progs/e25run.lsp"` или `"c:\\D_Progs\\e25run.lsp"`. В качестве разделителя каталогов может использоваться косая черта или обратная косая черта, но обратная черта должна обязательно удваиваться (см. замечание к строковым константам в *разд. 1.2.1*).

Если расширение в имени файла опущено, то AutoCAD пытается выполнить загрузку, поочередно добавляя следующие расширения: vlx, fas, lsp.

Расширение vlx соответствует файлам VLX-приложений, fas — компилированным LISP-файлам или файлам FAS-приложений, lsp — исходным LISP-файлам. Для непосредственного редактирования пользователю доступны только файлы с расширением lsp, т. к. файлы с расширениями vlx и fas не являются текстовыми. О компилированных файлах (с расширением fas) и файлах приложений (с расширениями fas и vlx) *см. главу 2.*

1.2.5. Функция *vl-load-all*

Система AutoCAD разрешает работу одновременно с несколькими открытыми документами (рисунками). Поэтому возникает проблема, загружен ли файл с расширением lsp в конкретный рисунок. Функция `load` выполняет загрузку только в текущий документ. Для того чтобы обеспечить загрузку LISP-файла во все уже открытые и открываемые в данном сеансе AutoCAD рисунки, следует пользоваться функцией `vl-load-all`:

```
(vl-load-all <файл>)
```

Аргумент `<файл>` — текстовая строка с именем файла (расширение файла может быть тем же, что и у первого аргумента функции `load`). Возвращаемого значения нет, если загрузка не выполнилась. Возвращается значение `T` в случае успешной загрузки.

См. также *разд. 1.17.29*.

1.3. Выполнение команд AutoCAD

Очень важной является функция `command`, которая имитирует ввод пользователя в командной строке AutoCAD с клавиатуры. Аналогичную роль играет функция `vl-cmdf` (различия между ними разбираются в *разд. 1.3.2*).

1.3.1. Функция *command*

Синтаксис функции:

```
(command [<параметр1> [<параметр2> ... [<параметрN>] ... ]])
```

Возвращаемое значение — `nil`.

Функция может вызываться без параметров — это равносильно нажатию клавиши <Esc>, что прерывает действие активной команды. Если параметры используются, то их количество может быть любым. Каждый из параметров чаще всего является текстовой строкой, имитирующей набор символов строки на клавиатуре с нажатием клавиши <Enter>.

Замечание

Если в качестве параметров функции `command` передаются выражения LISP, то среди этих выражений не должны фигурировать функции интерактивного ввода (`getint`, `getreal`, `getpoint` и т. д. — *см. разд. 1.10*). Если указанное ограничение было в программе нарушено, то результат операции непредсказуем. Следует вынести использование функций интерактивного ввода перед выражением с функцией `command`.

Предположим, необходимо нарисовать отрезками рамку формата размером 210 на 297 (левый нижний угол рамки взять в точке (0,0) и верхний правый — в точке

(210,297)). Вспомним, что команда рисования отрезков в AutoCAD называется LINE (ОТРЕЗОК). Если вы не уверены, что ваша программа всегда будет работать на компьютере с русской версией AutoCAD, то используйте английские имена команд с подчеркиванием (в данном случае — _LINE). С такими именами команды будут выполняться в любой версии системы. Кроме того, перед или после символа подчеркивания желательно ставить точку (_.LINE или ._LINE), если вы не хотите работать с переопределенной командой (а возможность переопределения стандартных команд в системе есть).

Если бы вы выполняли задачу без программы, то вводили с клавиатуры следующие команды и опции:

- `_.LINE`
- `0,0`
- `210,0`
- `210,297`
- `0,297`
- `_C`

По этим указаниям AutoCAD вызовет команду LINE, возьмет в качестве первой точки (0,0), в качестве второй — (210,0), в качестве третьей — (210,297), в качестве четвертой — (0,297), а `_C` (соответствует опции **Close** (Замкнуть)) замкнет отрезки на первую точку (опции тоже желательно вводить английские и с подчеркиванием). Программа, имитирующая описанный алгоритм, занимает одну строку и выглядит так, как показано в листинге 1.1.

Листинг 1.1. Построение прямоугольника (вариант 1)

```
; Глава 01\Листинг 1.01.lsp
(command "_.LINE" "0,0" "210,0" "210,297" "0,297" "_C")
```

Можно указанное построение выполнить в несколько строк (в несколько обращений к функции `command`, как в листинге 1.2).

Листинг 1.2. Построение прямоугольника (вариант 2)

```
; Глава 01\Листинг 1.02.lsp
(command "_.LINE" "0,0" "210,0")
(command "210,297" "0,297")
(command "_C")
```

В листинге 1.2 использовано трехкратное обращение к функции `command`, в каждом из которых количество параметров должно быть не меньше одного.

Можно координаты точек задавать не как строку, а как списки из двух чисел целого или вещественного типа (листинг 1.3).

Листинг 1.3. Построение прямоугольника (вариант 3)

```
; Глава 01\Листинг 1.03.lsp
(command "_.LINE" '(0 0) '(210 0) '(210 297) '(0 297) "_C")
```

Апостроф опускать нельзя, поскольку выражение `(210 0)` без апострофа будет рассматриваться как обращение к функции `210` с аргументом `0`.

Там, где требуется ввод номеров (чисел), можно в качестве аргументов передавать функции `command` именно числа, а не строки с числами, например:

```
(command "_.COLOR" "1")
(command "_.COLOR" 1)
```

Оба выражения выполняют одну и ту же операцию (в качестве текущего устанавливают красный цвет, который имеет номер 1), но в первом случае номер цвета задается как строка, а во втором — как целое число.

Можно в одной функции `command` обратиться сразу к нескольким командам системы AutoCAD, например:

```
(command "_.COLOR" 1 "_.ZOOM" "_E")
```

Эта программная строка выполнит команду COLOR (ЦВЕТ) для установки красного цвета и команду ZOOM (ПОКАЗАТЬ) с опцией **Extents** (Границы).

Для имитации варианта, когда пользователь вместо ввода опции нажимает клавишу <Enter>, в качестве параметра функции `command` выступает текстовая строка нулевой длины (`""`). Например, если бы вы в варианте, приведенном в листинге 1.1, захотели завершить команду LINE не опцией `_C` (замкнуть прямоугольник), а нажатием клавиши <Enter> (не замыкать прямоугольник), то соответствующая программная строка выглядела бы так:

```
(command "_.LINE" "0,0" "210,0" "210,297" "0,297" "")
```

1.3.2. Функция *vl-cmdf*

Функция `vl-cmdf` имеет следующий синтаксис:

```
(vl-cmdf [<параметр1> [<параметр2> ... [<параметрN>] ... ]])
```

Аргументами могут быть любые выражения, которые нужно передать в командную строку AutoCAD (в том числе команды, опции команд, данные, выражения LISP и т. д.). В отличие от функции `command`, в качестве параметров функции `vl-cmdf` могут фигурировать выражения LISP, содержащие функции интерактивного ввода (`getint` и т. п.).

Отсутствие аргументов функции равносильно нажатию пользователем клавиши <Esc>.

Возвращаемое значение — `T`, если все выражения, переданные в командную строку, выполнились без сбоев, и `nil` — при обнаружении ошибок.

Основное отличие функции `vl-cmdf` от функции `command` заключается в том, что проверка выполнимости команд AutoCAD с заданными параметрами осуществляется до реального начала их исполнения. И если обнаруживается какая-то ошибка, то ни один из параметров функции `vl-cmdf` не выполняется системой AutoCAD, и генерируется сообщение об ошибке приложения. Это исключает возможность частичной работы любых команд AutoCAD (т. е. когда выполняются первые операции, а затем команда выдает сбой).

Функция `command` по одному передает параметры в командную строку, и AutoCAD по одному пытается их выполнить.

В примерах, приведенных в листингах 1.1—1.3, функцию `command` можно было бы заменить на функцию `vl-cmdf`.

1.4. Арифметические функции

Самыми простыми являются *арифметические функции*, они рассматриваются в следующих разделах. Тип результата арифметической функции зависит от типов ее аргументов. Если все аргументы одного типа, то и результат будет того же типа. Если среди аргументов встречаются как целые, так и вещественные, то результат будет вещественным.

Для каждой функции указывается синтаксис, количество и типы аргументов, а также тип возвращаемого значения.

1.4.1. Функция +

Вычисляет сумму чисел.

```
(+ [<число1> [<число2> ... [<числоN>] ... ]])
```

Типы аргументов: целые или вещественные числа.

Тип возвращаемого значения: целое, если все аргументы целые; вещественное, если хотя бы один аргумент является вещественным.

Если количество аргументов равно нулю, то возвращается `0` (целое). Если количество аргументов равно 1, то возвращается значение, равное значению аргумента.

Примеры:

- `(+ 16 30)` возвращает `46`
- `(+)` возвращает `0`
- `(+ -2.0 12.1)` возвращает `10.1`
- `(+ 2 13 1.00)` возвращает `16.0`

1.4.2. Функция –

Вычисляет разность чисел.

```
(- [<число1> [<число2> ... [<числоN>] ... ]])
```

Типы аргументов: целые или вещественные числа.

Тип возвращаемого значения: целое, если все аргументы целые; вещественное, если хотя бы один аргумент является вещественным.

Если аргументы не заданы, то возвращается `0` (целое). Если количество аргументов равно 1, то возвращается значение, вычитаемое из 0 (т. е. `<число1>` с противоположным знаком). Если количество аргументов равно 2 и более, то из первого числа вычитается сумма всех остальных.

Примеры:

- `(- 3)` возвращает `-3`
- `(-)` возвращает `0`
- `(- -4 2.1 2.3)` возвращает `-8.4`

1.4.3. Функция *

Вычисляет произведение чисел.

```
(* [<число1> [<число2> ... [<числоN>] ... ]])
```

Типы аргументов: целые или вещественные числа.

Тип возвращаемого значения: целое, если все аргументы целые; вещественное, если хотя бы один аргумент является вещественным.

Если количество аргументов равно нулю, то возвращается `0` (целое). Если количество аргументов равно 1, то возвращается значение аргумента. Если количество аргументов равно 2 и более, то выполняется перемножение всех аргументов.

Примеры:

- `(* 7)` возвращает `7`
- `(*)` возвращает `0`
- `(* 5 1.1 -0.3)` возвращает `-1.65`

Замечание

При перемножении больших целых чисел следует учитывать, что произведение может выйти за границы допустимых значений и функция выдаст непредсказуемый результат. Так, выражение `(* 12345 987654)` вернет значение `-692313258`, что заведомо не может быть правильным, т. к. произведение положительных чисел должно быть положительным. В таких случаях числа следует перемножать в вещественном виде.

1.4.4. Функция /

Вычисляет частное чисел.

```
(/ [<число1> [<число2> ... [<числоN>] ... ]])
```

Типы аргументов: целые или вещественные числа.

Тип возвращаемого значения: целое, если все аргументы целые; вещественное, если хотя бы один аргумент является вещественным.

Если количество аргументов равно нулю, то возвращается `0` (целое). Если количество аргументов равно 1, то возвращается значение аргумента. Если количество аргументов равно 2 и более, то выполняется деление первого аргумента на произведение всех остальных.

Примеры:

- `(/ 81)` возвращает `81`
- `(/)` возвращает `0`
- `(/ 5 1.1 -0.3)` возвращает `-15.1515` (реальный результат будет получен с максимальной точностью, но на экран обычно выводятся шесть значащих цифр)
- `(/ 17 23)` возвращает `0`
- `(/ -17 23)` возвращает `0`

Замечание

При делении целых чисел результат должен быть того же типа, что и аргументы, т. е. целым, поэтому он усекается до целого (путем отбрасывания дробной части). Этим объясняется, что выражения `(/ 17 23)` и `(/ -17 23)` возвращают значение `0`.

1.4.5. Функция *1+*

Увеличивает аргумент на 1.

```
(1+ <число>)
```

Тип аргумента: целое или вещественное число.

Тип возвращаемого значения: целое, если аргумент целый; вещественное, если аргумент является вещественным.

Примеры:

- `(1+ 2)` возвращает `3`
- `(1+ 5.2)` возвращает `6.2`
- `(1+ -5.2)` возвращает `-4.2`

1.4.6. Функция *1–*

Уменьшает аргумент на 1.

```
(1- <число>)
```

Тип аргумента: целое или вещественное число.

Тип возвращаемого значения: целое, если аргумент целый; вещественное, если аргумент является вещественным.

Примеры:

- `(1- 11)` возвращает `10`
- `(1- 1.17)` возвращает `0.17`

1.4.7. Функция ~

Вычисляет дополнение числа до –1 (т. е. сумма возвращаемого значения функции и аргумента `<целое>` равна –1).

```
(~ <целое>)
```

Тип аргумента: целое число.

Тип возвращаемого значения: целое.

Примеры:

- `(~ 9)` возвращает `-10`
- `(~ 0)` возвращает `-1`

1.4.8. Функция *abs*

Вычисляет абсолютную величину числа.

```
(abs <число>)
```

Тип аргумента: целое или вещественное число.

Тип возвращаемого значения: целое, если аргумент целый; вещественное, если аргумент является вещественным.

Примеры:

- `(abs 3)` возвращает `3`
- `(abs -1.1)` возвращает `1.1`
- `(abs 0.0)` возвращает `0.0`

1.5. Логические функции

В следующих разделах рассматриваются функции логических и условных операций и проверок. Здесь же приводятся функции `while` и `repeat`, позволяющие организовывать циклы.

1.5.1. Функция =

Проверяет равенство аргументов (рекомендуется использовать для чисел или строк).

```
(= <аргумент1> [<аргумент2> ... [<аргументN>]...])
```

Типы аргументов: любые.

Тип возвращаемого значения: логическое (`T`, если значения аргументов совпали, и `nil`, если не совпали).

Если задан только один аргумент, то возвращается `T`. Если количество аргументов 2 или больше, то при совпадении значений всех аргументов возвращается `T`, в противном случае — `nil`. При сравнении строк `T` выдается только в случае, когда длины всех строк равны и все строки посимвольно совпадают. При сравнении чисел `T` выдается также в том случае, когда одни аргументы целые, а другие вещественные, но их значения совпадают.

Примеры:

- `(= -17.0 -17 -17.000)` возвращает `T`
- `(= 1)` возвращает `T`
- `(= "33")` возвращает `T`
- `(= nil)` возвращает `T` (потому что один аргумент)
- `(= 33 "33")` возвращает `nil`
- `(= nil nil)` возвращает `T`
- `(= nil T)` возвращает `nil`
- `(= T T)` возвращает `T`
- `(= T 1)` возвращает `nil`
- `(= 2 30)` возвращает `nil`
- `(= "AP8" "Ap8" "AP8")` возвращает `nil`
- `(= "2abc" "2abc")` возвращает `T`

См. также функции `eq` и `equal`.

1.5.2. Функция /=

Проверяет неравенство аргументов (рекомендуется использовать для чисел или строк).

```
(/= <аргумент1> [<аргумент2> ... [<аргументN>]...])
```

Типы аргументов: любые.

Тип возвращаемого значения: логическое (`T`, если значения каких-то аргументов не совпали, и `nil`, если совпали). Если задан только один аргумент, то возвращается `T`.

Примеры:

- `(/= 5)` возвращает `T`
- `(/= -2.0 -2)` возвращает `nil`
- `(/= 32 30)` возвращает `T`
- `(/= "AP8" "Ap8" "AP8")` возвращает `T`
- `(/= "AP8" "AP8" "AP8")` возвращает `nil`
- `(/= "2abc" "2abc")` возвращает `nil`

1.5.3. Функция <

Проверяет расположение аргументов по возрастанию, слева направо.

```
(< <аргумент1> [<аргумент2> ... [<аргументN>]...])
```

Типы аргументов: числа (целые, вещественные) или строки.

Тип возвращаемого значения: логическое (`T`, если значения аргументов расположены по возрастанию, и `nil` — в противном случае). Для текстовых строк производится посимвольное сравнение, по возрастанию кодов символов[1].

Если задан только один аргумент, то возвращается `T`.

Примеры:

- `(< -1)` возвращает `T`
- `(< -5.81 0)` возвращает `T`
- `(< 12 12 20)` возвращает `nil`
- `(< "X" "Y" "Z")` возвращает `T`
- `(< "2abc" "2abc")` возвращает `nil`
- `(< "d" "d5")` возвращает `T`
- `(< "d5" "d")` возвращает `nil`

---

[1] Имеются в виду символы текущей кодовой таблицы. *См. функцию* `ascii`.

1.5.4. Функция <=

Проверяет расположение аргументов по неубыванию, слева направо.

```
(<= <аргумент1> [<аргумент2> ... [<аргументN>]...])
```

Типы аргументов: числа (целые, вещественные) или строки.

Тип возвращаемого значения: логическое (`T`, если значения аргументов расположены по неубыванию, и `nil` — в противном случае). Для текстовых строк производится посимвольное сравнение, по неубыванию кодов символов.

Если задан только один аргумент, то возвращается `T`.

Примеры:

- `(<= -1)` возвращает `T`
- `(<= -5.81 0)` возвращает `T`
- `(<= 12 12 20)` возвращает `T`
- `(<= "X" "Y" "Z")` возвращает `T`
- `(<= "2abc" "2abc")` возвращает `nil`
- `(<= "d" "d5")` возвращает `T`
- `(<= "d5" "d")` возвращает `nil`

1.5.5. Функция >

Проверяет расположение аргументов по убыванию, слева направо.

```
(> <аргумент1> [<аргумент2> ... [<аргументN>]...])
```

Типы аргументов: числа (целые, вещественные) или строки.

Тип возвращаемого значения: логическое (`T`, если значения аргументов расположены по убыванию, и `nil` — в противном случае). Для текстовых строк производится посимвольное сравнение, по убыванию кодов символов.

Если задан только один аргумент, то возвращается `T`.

Примеры:

- `(> -1)` возвращает `T`
- `(> -5.81 0)` возвращает `nil`
- `(> 12 12 20)` возвращает `nil`
- `(> "X" "Y" "Z")` возвращает `nil`
- `(> "2abc" "2abc")` возвращает `nil`
- `(> "d" "d5")` возвращает `nil`
- `(> "d5" "d")` возвращает `T`

1.5.6. Функция >=

Проверяет расположение аргументов по невозрастанию, слева направо.

```
(>= <аргумент1> [<аргумент2> ... [<аргументN>]...])
```

Типы аргументов: числа (целые, вещественные) или строки.

Тип возвращаемого значения: логическое (`T`, если значения аргументов расположены по невозрастанию, и `nil` — в противном случае). Для текстовых строк производится посимвольное сравнение, по невозрастанию кодов символов.

Если задан только один аргумент, то возвращается `T`.

Примеры:

- `(>= -1)` возвращает `T`
- `(>= -5.81 0)` возвращает `nil`
- `(>= 12 12 20)` возвращает `nil`
- `(>= "X" "Y" "Z")` возвращает `nil`
- `(>= "2abc" "2abc")` возвращает `T`
- `(>= "d" "d5")` возвращает `nil`
- `(>= "d5" "d")` возвращает `T`

1.5.7. Функция *and*

Выполняет операцию логического И.

```
(and [<аргумент1> [<аргумент2> ... [<аргументN>]...]])
```

Типы аргументов: любые.

Тип возвращаемого значения: логическое (`T`, если значения всех аргументов истинны, и `nil` — в противном случае). К истинным, помимо `T`, относятся все значения (числовые, строковые и т. д.), кроме `nil`.

Если аргументы не заданы или задан только один аргумент, то возвращается `T`.

Примеры:

- `(and 11)` возвращает `T`
- `(and T nil T)` возвращает `nil`
- `(and T T 4)` возвращает `T`
- `(and "x" "y" "z")` возвращает `T`

1.5.8. Функция *or*

Выполняет операцию логического ИЛИ.

```
(or [<аргумент1> [<аргумент2> ... [<аргументN>]...]])
```

Типы аргументов: любые.

Тип возвращаемого значения: логическое (`T`, если значение хотя бы одного аргумента истинно, и `nil` — в противном случае). К истинным, помимо `T`, относятся все значения (числовые, строковые и т. д.), кроме `nil`.

Если аргументы не заданы или задан только один аргумент, то возвращается `T`.

Примеры:

- `(or 11)` возвращает `T`
- `(or T nil T)` возвращает `T`
- `(or T T 4)` возвращает `T`
- `(or "x" "y" "z")` возвращает `T`
- `(or nil nil nil)` возвращает `nil`

1.5.9. Функция *not*

Выполняет операцию логического отрицания.

```
(not <аргумент>)
```

Типы аргументов: любые.

Тип возвращаемого значения: логическое (`T`, если значение аргумента равно `nil`, и `nil` — в противном случае).

Примеры:

- `(not T)` возвращает `nil`
- `(not nil)` возвращает `T`
- `(not ".NET")` возвращает `nil`
- `(not -64.64)` возвращает `nil`

1.5.10. Функция *null*

Выполняет операцию проверки на `nil`.

```
(null <аргумент>)
```

Типы аргументов: любые.

Тип возвращаемого значения: логическое (`T`, если значение аргумента равно `nil`, и `nil` — в противном случае).

Фактически идентична функции `not`. Традиционно функция `null` употребляется для проверки непустоты списков (поскольку пустой список `'()` идентичен `nil`) или функций, а `not` — для аргументов других типов.

Примеры:

- `(null T)` возвращает `nil`
- `(null nil)` возвращает `T`
- `(null '())` возвращает `T`
- `(null ".NET")` возвращает `nil`

1.5.11. Функция *if*

Выполняет условную операцию типа if-then-else (если-то-иначе).

```
(if <условие> <выражение1> [<выражение2>])
```

Типы аргументов: любые.

Возвращаемое значение: вычисленное значение аргумента `<выражение1>`, если значение аргумента `<условие>` отлично от `nil` (напомним, что `nil` в логических операциях используется как "ложь"); значение аргумента `<выражение2>`, если значение аргумента `<условие>` равно `nil` (причем когда `<выражение2>` опущено, тогда возвращается `nil`). Используется как условный оператор.

Примеры:

- `(if (> a b) 2 8)` возвращает `2`, если `a>b`, и `8` — в других случаях
- `(if (= x1 x2) (+ x1 102) (* x1 3.04))` возвращает результат вычисления выражения `(+ x1 102)`, если значения `x1` и `x2` равны; иначе — результат вычисления выражения `(* x1 3.04)`
- `(if (< a b) "a<b")` возвращает строку `"a<b"` или `nil` (т. к. `<выражение2>` опущено)
- `(if nil "Initialization")` возвращает `nil` (т. к. `<условие>` всегда ложно)

Если функция `if` используется для выполнения по условию не одного выражения, а нескольких, то эти выражения следует объединить с помощью рассматриваемой в следующем разделе функции `progn` (иначе произойдет синтаксическая ошибка, вызванная неправильным количеством аргументов при обращении к функции `if`).

1.5.12. Функция *progn*

Объединяет несколько (не менее двух) выражений в одно, когда по синтаксису LISP может использоваться только одно (как в функции `if`).

```
(progn <выражение1> <выражение2> ... [<выражениеN>])
```

Типы аргументов: любые.

Возвращаемое значение — значение последнего из объединенных выражений.

Рассмотрим использование функции `progn` на таком примере. Пусть при истинности условия `(> a b)` нужно выполнить выражения `(setq c -1)`, `(setq d (- c a b))` и `(* d b)`, а при ложности условия — рассчитать выражения `(setq c 2)`, `(setq d (* a b))` и `(* d c)`. Программный код приведен в листинге 1.4.

Листинг 1.4. Пример совместного использования функций `if` и `progn`

```
; Глава 01\Листинг 1.04.lsp
(if (> a b)
  (progn
    (setq c -1)
    (setq d (- c a b))
    (* d b)
  ); конец progn
  (progn
    (setq c 2)
    (setq d (* a b))
    (* d c)
  ); конец progn
); конец if
```

Для удобства чтения в этом фрагменте текста программы в начале некоторых строк добавлены пробелы, чтобы открывающая скобка оказалась на одном уровне с закрывающей, а внутренние выражения были смещены относительно ближайших внешних на две позиции вправо. К закрывающим скобкам даны комментарии, согласно правилам их написания *(см. разд. 1.1.3)*.

1.5.13. Функция *cond*

Условная операция (типа `case` в языках программирования высокого уровня) с любым количеством условий.

```
(cond [(<условие1> [<выражение11> ... ])[(<условие2>[<выражение21> ... ])] ...])
```

Аргументы: любое количество списков, в каждом из которых на первом месте стоит проверяемое условие, а затем следуют выражения, подлежащие выполнению (вычислению), если это условие оказалось истинным.

Возвращаемое значение: последнее вычисленное выражение того списка, условие в котором оказалось истинным (если таких списков несколько, то выполняются только выражения первого списка с истинным условием). Если вычисляемое выражение отсутствует (не задано в списке с истинным условием), то возвращается результат вычисления аргумента `<условие>`. Если аргументы функции `cond` не заданы или если условия во всех списках оказались ложными, то возвращается `nil`.

Примеры:

- `(cond)` возвращает `nil`
- `(cond (nil 55))` возвращает `nil`
- `(cond (T 55))` возвращает `55`
- `(cond ((= 5 6) 55) ((= 7 19) 97))` возвращает `nil`
- `(cond ((= 2 2) 100) (T 500000))` возвращает `100`
- `(cond ((not nil) 534))` возвращает `534`
- `(cond (80))` возвращает `80`

Впрочем, использование функции `cond` с неполным количеством аргументов не имеет особого смысла.

Рассмотрим задачу: переменной `mycolor` нужно присвоить текстовое значение с наименованием одного из основных цветов AutoCAD (`"красный"`, `"желтый"`, `"зеленый"`, `"голубой"`, `"синий"`) в зависимости от текущего значения переменной `cc`, которая принимает значения 1, 2, 3, 4, 5. Если значение переменной `cc` оказалось отличным от перечисленных, то переменной `mycolor` нужно присвоить `""` (пустая строка). Программный код приведен в листинге 1.5.

Листинг 1.5. Пример 1 использования функции `cond`

```
; Глава 01\Листинг 1.05.lsp
(setq mycolor
  (cond
    ((= cc 1) "красный")
    ((= cc 2) "желтый")
    ((= cc 3) "зеленый")
    ((= cc 4) "голубой")
    ((= cc 5) "синий")
    (T "")
  ); конец cond
); конец setq
```

Функция `setq` присваивает переменной `mycolor` результат вычисления функции `cond`. Значение, возвращаемое функцией `cond`, вычисляется по следующей схеме. Сначала рассматривается список, заданный в качестве первого аргумента — `((= cc 1) "красный")` — и проверяется условие, являющееся первым элементом этого списка. Если оказалось, что значение `cc` равно 1 и, значит, первое условие возвращает "истину", то дальнейшее рассмотрение аргументов функции `cond` не выполняется, а в качестве возвращаемого значения принимается `"красный"`. Если первое условие не вычислилось как "истина", то проверяется второе условие и т. д. В качестве последнего, шестого, условия стоит `T` (т. е. "истина" при любых значениях), поэтому если значение переменной `cc` не совпало с 1, 2, 3, 4, 5, то

в качестве возвращаемого значения функции cond будет принято "" (пустая строка).

Еще один пример иллюстрирует случай, когда по истинности значения аргумента <условие> выполняются несколько операторов (листинг 1.6).

Листинг 1.6. Пример 2 использования функции cond

```
; Глава 01\Листинг 1.06.lsp
(cond
  ((= cc 1) (setq abc "красный")
            (command "_.PLINE" "-50,120.45" "97.66,23.124" "45.7,800"
                                "_C")
  ); cc равно 1
  ((= cc 2) (setq abc "желтый")
            (command "_.CIRCLE" "-50,120.45" "100")
  ); cc равно 2
); конец cond
```

В этом примере функция cond проверяет значение переменной cc. Если оно равно 1, то функция setq присваивает переменной abc значение "красный", а затем AutoCAD рисует замкнутую полилинию с тремя вершинами. Если значение cc равно 2, то функция setq присваивает переменной abc значение "желтый", а AutoCAD рисует окружность радиуса 100. Если окажется, что значение cc не равно ни 1, ни 2, то функция cond вернет nil, не изменяя при этом значения переменной abc и не строя никаких новых примитивов AutoCAD. Поскольку функция command является последним выражением как в первом, так и во втором условиях функции cond, то во всех трех случаях будет возвращено значение nil (nil является возвращаемым значением функции command — *см. разд. 1.3.1*). Однако в данном примере функция cond применена не ради возвращаемого значения, а для изменения значения переменной abc и построения дополнительной полилинии или окружности.

1.5.14. Функция *minusp*

Проверка числа на отрицательность.

(minusp <число>)

Тип аргумента: целое или вещественное число.

Возвращаемое значение: T, если <число> отрицательно, и nil — в противном случае.

Примеры:

- (minusp 0) возвращает nil
- (minusp -38.08) возвращает T
- (minusp 710.2) возвращает nil

1.5.15. Функция *zerop*

Проверка числа на нуль.

```
(zerop <число>)
```

Тип аргумента: целое или вещественное число.

Возвращаемое значение: `T`, если `<число>` равно нулю, и `nil` — в противном случае.

Примеры:

- `(zerop 0)` возвращает `T`
- `(zerop 0.0)` возвращает `T`
- `(zerop 64)` возвращает `nil`
- `(zerop -4.1)` возвращает `nil`

1.5.16. Функция *numberp*

Проверка типа аргумента на числовой.

```
(numberp <аргумент>)
```

Тип аргумента: любой.

Возвращаемое значение: `T`, если `<аргумент>` является вещественным или целым числом, и `nil` — в противном случае.

Примеры:

- `(numberp 0)` возвращает `T`
- `(numberp nil)` возвращает `nil`
- `(numberp T)` возвращает `nil`
- `(numberp -220.0)` возвращает `T`
- `(numberp "95")` возвращает `nil`
- `(numberp '(14.39))` возвращает `nil` (список не относится к числам)

1.5.17. Функция *listp*

Проверка принадлежности аргумента (по типу) к спискам, точечным парам или точечным спискам.

```
(listp <аргумент>)
```

Тип аргумента: любой.

Возвращаемое значение: `T`, если `<аргумент>` является списком, точечной парой, точечным списком или `nil` (т. к. пустой список `'()` вырождается в `nil`), и `nil` — в противном случае.

Примеры:

- `(listp 0)` возвращает `nil`
- `(listp 70.2e+23)` возвращает `nil`
- `(listp "as")` возвращает `nil`
- `(listp '(-41.44))` возвращает `T`
- `(listp '(8 . "Upper"))` возвращает `T`
- `(listp '(3 5 6 . 4))` возвращает `T`
- `(listp nil)` возвращает `T`

1.5.18. Функция *vl-consp*

Проверяет, является ли `<аргумент>` непустым списком, непустой точечной парой или непустым точечным списком (сравните с функцией `listp`).

```
(vl-consp <аргумент>)
```

Возвращаемое значение — `T` или `nil`.

Примеры:

- `(vl-consp '(1.3 3.1))` возвращает `T`
- `(vl-consp '(8 . "Upper"))` возвращает `T`
- `(vl-consp '(3 5 6 . 4))` возвращает `T`
- `(vl-consp '())` возвращает `nil`
- `(vl-consp 125)` возвращает `nil`

1.5.19. Функция *atom*

Проверяет, является ли `<аргумент>` атомом (т. е. не является непустым списком, а также непустой точечной парой или непустым точечным списком).

```
(atom <аргумент>)
```

Возвращаемое значение — `T` или `nil`.

Примеры:

- `(atom nil)` возвращает `T`
- `(atom '())` возвращает `T` (поскольку `nil` и `'()` — одно и то же)
- `(atom '(3 99 3))` возвращает `nil`
- `(atom '(62 . 7))` возвращает `nil`
- `(atom 0)` возвращает `T`
- `(atom "0")` возвращает `T`

1.5.20. Функция *vl-symbolp*

Проверяет, является ли `<аргумент>` символом (указателем функции или переменной).

```
(vl-symbolp <аргумент>)
```

Возвращаемое значение — `T` или `nil`.

Примеры:

- `(vl-symbolp T)` возвращает `T`
- `(vl-symbolp 'T)` возвращает `T`
- `(vl-symbolp nil)` возвращает `nil`
- `(vl-symbolp 'nil)` возвращает `nil`
- `(vl-symbolp 1)` возвращает `nil`
- `(vl-symbolp 'command)` возвращает `T`
- `(vl-symbolp '1+)` возвращает `T`

1.5.21. Функция *boundp*

Проверка, присвоено ли значение символу.

```
(boundp <аргумент>)
```

Тип аргумента: любой.

Возвращаемое значение: `T`, если `<аргумент>` указывает на символ, имеющий значение (но не `nil`), и `nil`, если значение не задано или равно `nil` (для LISP это одно и то же). Для указания на имя функции или имя переменной его следует предварять апострофом.

Примеры:

- `(boundp '+)` возвращает `T`
- `(boundp 'cos)` возвращает `T`
- `(boundp 'pi)` возвращает `T`
- `(boundp "M")` возвращает `nil`

1.5.22. Функция *eq*

Проверяет, равны ли значения двух аргументов (аналогично функции =) или указывают ли они на один и тот же символ.

```
(eq <аргумент1> <аргумент2>)
```

Типы аргументов: любые.

Тип возвращаемого значения: логическое (`T`, если значения аргументов совпали или они указывают на один и тот же символ, и `nil` — в противном случае).

Примеры:

- `(eq -17.0 -17)` возвращает `T`
- `(eq 4 3)` возвращает `nil`
- `(eq pi pi)` возвращает `T`
- `(eq 'pi 'pi)` возвращает `T`
- `(eq "кла" "кла")` возвращает `T`
- `(eq "HNY" "hnY")` возвращает `nil`

1.5.23. Функция *equal*

Проверяет равенство двух объектов (для чисел и списков из числовых величин сравнение производится в пределах допуска).

```
(equal <аргумент1> <аргумент2> [<допуск>])
```

Типы аргументов: любые.

Тип возвращаемого значения: логическое (`T`, если абсолютная величина разности аргументов не превосходит значения аргумента `<допуск>`, и `nil`, если не совпали; когда `<допуск>` не задан, то его значение считается равным нулю). Для аргументов, не являющихся числами или списками из чисел, `<допуск>` не используется.

Примеры:

- `(equal -17.0 -17)` возвращает `T`
- `(equal 6.000 6.005 0.005)` возвращает `T`
- `(equal "cow" "cow")` возвращает `T`
- `(equal '(3.00 54.2 0.99) '(3 54.2 1.0) 0.01)` возвращает `nil` (на границе допуска функция может давать неверный результат)
- `(equal '(3.00 54.2 0.99) '(3 54.2 1.0) 0.011)` возвращает `T`

1.5.24. Функция *while*

Выполняет операцию цикла по многократно проверяемому условию.

```
(while <условие> [<выражение1> [<выражение2> ... [<выражениеN>]...]])
```

Типы аргументов: любые.

Функция `while` на каждом шаге цикла вычисляет значение аргумента `<условие>` и, если это значение "истинно" (т. е. не `nil`), выполняет (вычисляет) следующие за условием выражения. Функция прекратит свою работу, когда на некотором шаге

аргумент <условие> получит значение nil. Аргумент <условие> должен быть таким, чтобы на каком-то шаге он получил бы значение nil, иначе цикл, организуемый с помощью функции while, станет бесконечным (программа "зациклится").

Возвращаемое значение: значение аргумента <выражениеN> на том шаге, когда последнее вычисление аргумента <условие> выдало значение, отличное от nil. Если уже на первом шаге <условие> имеет значение nil, то функция while ничего не выполняет и возвращает nil.

Рассмотрим следующий пример. Пусть надо вычислить значение n! (факториал числа n), т. е. произведение целых чисел от 1 до n. В данном подразделе восклицательный знак используется как знак функции факториала. Программный код приведен в листинге 1.7.

Листинг 1.7. Пример использования функции while (вычисление факториала)

```
; Глава 01\Листинг 1.07.lsp
;;; (setq n 10) ; Задайте n
(setq i 1 factorial 1)
(while (< i n)
  (setq i (1+ i))
  (setq factorial (* factorial i))
); конец while
```

Рассмотрим работу примера, когда число n равно 10 (т. е. вычисляется 10!). Для этого необходимо раскомментировать вторую строку листинга.

В программе используются переменные i (является счетчиком цикла) и factorial (накапливает произведение чисел, формирующее факториал). Перед входом в цикл они получают начальные значения — соответственно, 1 и 1 (вспомните, что по определению 1! считается равным 1).

Функция while проверяет условие (< i n) для текущего значения i, и если результат вычисления условия отличен от nil, то выполняет внутренние операции (две операции с участием функции setq). Таким образом, на первом шаге цикла i равно 1, проверяемое условие (< 1 n) возвращает значение T ("истина"), поэтому функция while увеличивает i на 1 (получается i = 2) и умножает переменную factorial на i: factorial = 1*2 (не что иное, как 2!).

Далее снова передается управление на вход в цикл (уже при i = 2). Проверка условия опять дает результат "истина", поэтому i получает значение 3 (2+1), а factorial — 6 (2*3). И так далее, пока i не станет равным n (10). Тогда программа покинет цикл, не выполняя внутренних операций. Результат: при n = 10 factorial = 3 628 800.

1.5.25. Функция *repeat*

Выполняет операцию цикла с фиксированным количеством повторений.

```
(repeat <количество> [<выражение1> ...])
```

Типы аргументов: <количество> — целое число (имеют смысл только положительные числа), <выражение1> — любое выражение. После аргумента <выражение1> могут идти другие выражения, которые нужно выполнить внутри цикла.

Возвращаемое значение: значение последнего вычисленного выражения. Если <количество> имеет значение 0 или отрицательно или после аргумента <количество> не заданы выражения, то функция `repeat` возвращает `nil`.

Переработаем предыдущий пример (см. листинг 1.7) — вместо `while` воспользуемся функцией `repeat` (листинг 1.8).

Листинг 1.8. Пример использования функции `repeat` (вычисление факториала)

```
; Глава 01\Листинг 1.08.lsp
;;; (setq n 10) ; Задайте n
(setq i 1 factorial 1)
(repeat (1- n)
  (setq i (1+ i))
  (setq factorial (* factorial i))
); конец repeat
```

Поскольку входные значения `i` = 1 и `factorial` = 1! =1, то остается умножить `factorial` на 2, 3, ..., `n`. Количество таких умножений равно `n–1`, что на языке LISP записывается как `(1- n)`. Остальное работает, как в предыдущем примере.

1.6. Функции вычислений

Вычислительные функции также достаточно широко представлены в языке LISP.

1.6.1. Функция *fix*

Выполняет операцию усечения числа до целого (отбрасывается дробная часть).

```
(fix <число>)
```

Тип аргумента: целое или вещественное число.

Тип возвращаемого значения: целое.

Примеры:

- `(fix -24.51)` возвращает `-24`
- `(fix 87)` возвращает `87`

- `(fix 0.0)` возвращает `0`
- `(fix 39.123)` возвращает `39`

1.6.2. Функция *rem*

Получает остаток от деления аргумента `<число1>` на `<число2>`, затем остаток от деления результата на `<число3>` и т. д.

```
(rem [<число1> [<число2> ... [<числоN>]...]])
```

Типы аргументов: целые или вещественные числа.

Тип возвращаемого значения: целое, если все аргументы целые, и вещественное, если хотя бы один из аргументов вещественный. Знак результата определяется знаком первого аргумента (нулевое значение выводится без знака).

Если аргументы не заданы, то возвращается `0`. Если задан только аргумент `<число1>`, то возвращается его значение.

Примеры:

- `(rem)` возвращает `0`
- `(rem 3.3)` возвращает `3.3`
- `(rem 204 2)` возвращает `0`
- `(rem 7 3.2)` возвращает `0.6`
- `(rem 8.9 1.54)` возвращает `1.2`
- `(rem 49 11 3)` возвращает `2`
- `(rem -10 -3)` возвращает `-1`
- `(rem 10 -3)` возвращает `1`
- `(rem -10 3)` возвращает `-1`

1.6.3. Функция *max*

Определяет максимальное из чисел.

```
(max [<число1> [<число2> ... [<числоN>] ... ]])
```

Типы аргументов: целые или вещественные числа.

Тип возвращаемого значения: целое, если все аргументы целые, и вещественное, если хотя бы один из аргументов вещественный.

Если аргументы не заданы, то возвращается `0`. Если задан только один аргумент `<число1>`, то возвращается его значение.

Примеры:

- `(max 0 -32 -166)` возвращает `0`
- `(max 47 23.5 4.38027)` возвращает `47.0`

1.6.4. Функция *min*

Определяет минимальное из чисел.

```
(min [<число1> [<число2> ... [<числоN>] ... ]])
```

Типы аргументов: целые или вещественные числа.

Тип возвращаемого значения: целое, если все аргументы целые, и вещественное, если хотя бы один из аргументов вещественный.

Если аргументы не заданы, то возвращается `0`. Если задан только аргумент `<число1>`, то возвращается его значение.

Примеры:

- `(min 0 -32 -166)` возвращает `-166`
- `(min 47 23.5 4.38027)` возвращает `4.38027`

1.6.5. Функция *gcd*

Вычисляет наибольший общий делитель для двух положительных целых чисел.

```
(gcd <число1> <число2>)
```

Типы аргументов: целые числа.

Тип возвращаемого значения: целое.

Примеры:

- `(gcd 51 187)` возвращает `13`
- `(gcd 72 22)` возвращает `2`

1.6.6. Функция *angle*

Вычисляет угол между осью X текущей системы координат и вектором, идущим из первой точки, заданной аргументом `<точка1>`, во вторую точку, заданную аргументом `<точка2>`. Если точки трехмерные, то функция вычисляет угол между осью X и проекцией вектора, идущего из первой точки во вторую, на текущую плоскость построений.

```
(angle <точка1> <точка2>)
```

Типы аргументов: списки из двух или трех чисел (т. е. двумерные или трехмерные точки).

Возвращаемое значение — угол в радианах.

Примеры:

- `(angle '(0 0) '(125.0 0.0))` возвращает `0.0`
- `(angle '(30 0) '(30 45.0))` возвращает `1.5708` (AutoCAD показывает на экране в качестве возвращаемого значения `1.5708`, но на самом деле результат

будет с максимально возможной точностью — 15 знаков после десятичной точки)

- `(angle '(10 0) '(25.4 15.4))` возвращает `0.785398`

1.6.7. Функция *distance*

Вычисляет расстояние между первой точкой, заданной аргументом `<точка1>`, и второй точкой, заданной аргументом `<точка2>`. Если точки трехмерные, то функция рассчитывает угол между осью X и проекцией вектора, идущего из первой точки во вторую точку, на текущую плоскость построений.

```
(distance <точка1> <точка2>)
```

Типы аргументов: списки из двух или трех чисел (т. е. двумерные или трехмерные точки).

Возвращаемое значение — вещественное число. Если обе точки или хотя бы одна из них двумерная (третья координата оставшейся точки игнорируется), то возвращается двумерное расстояние между точками. Если обе точки трехмерные, то вычисляется расстояние в трехмерном пространстве.

Примеры:

- `(distance '(0 0) '(112.7 0.0))` возвращает `112.7`
- `(distance '(-3.1 -72.10 81.42) '(35.5 104.9 -24.0))` возвращает `209.6`

1.6.8. Функция *sqrt*

Вычисляет квадратный корень.

```
(sqrt <число>)
```

Тип аргумента: неотрицательное целое или вещественное число.

Возвращаемое значение — вещественное число.

Примеры:

- `(sqrt 49)` возвращает `7.0`
- `(sqrt 81.93)` возвращает `9.05152`

1.6.9. Функция *sin*

Вычисляет синус угла.

```
(sin <число>)
```

Тип аргумента: целое или вещественное число (угол в радианах).

Возвращаемое значение — вещественное число.

Примеры:

- `(sin -1)` возвращает `-0.841471`
- `(sin pi)` возвращает `1.22461e-016` (практически 0)
- `(sin 0.0)` возвращает `0.0`

1.6.10. Функция *cos*

Вычисляет косинус угла.

```
(cos <число>)
```

Тип аргумента: целое или вещественное число (угол в радианах).

Возвращаемое значение — вещественное число.

Примеры:

- `(cos -1.0)` возвращает `0.540302`
- `(cos 0)` возвращает `1.0`
- `(cos (* pi 0.5))` возвращает `6.12303e-017` (практически 0)

1.6.11. Функция *atan*

Вычисляет арктангенс.

```
(atan <число1> [<число2>])
```

Тип аргумента: целое или вещественное число.

Возвращаемое значение — вещественное число (угол в радианах).

Если второй аргумент опущен, то вычисляется арктангенс аргумента `<число1>`. Если заданы оба аргумента, то вычисляется арктангенс частного от деления первого аргумента на второй (`<число1>/<число2>`). Если `<число2>` равно нулю, то возвращается угол $+\pi/2$ или $-\pi/2$ (знак выбирается по знаку аргумента `<число1>`).

Примеры:

- `(atan 1)` возвращает `0.785398`
- `(atan -1 0)` возвращает `-1.5708`
- `(atan 2.3 4.7)` возвращает `0.455101`

1.6.12. Функция *exp*

Вычисляет натуральный антилогарифм — результат возведения в степень постоянной `e` (2.71828…).

```
(exp <число>)
```

Тип аргумента: целое или вещественное число.

Возвращаемое значение — вещественное число.

Примеры:

- `(exp 1.0)` возвращает `2.71828`
- `(exp 0)` возвращает `1.0`
- `(exp pi)` возвращает `23.1407`

1.6.13. Функция *expt*

Возводит число в степень.

```
(expt <основание> <степень>)
```

Типы аргументов: целые или вещественные числа.

Тип возвращаемого значения: целое, если оба аргумента целые, и вещественное, если хотя бы один аргумент — вещественное число. Если `<степень>` не является целым числом, то `<основание>` должно быть положительным.

Примеры:

- `(expt 3 2)` возвращает `9`
- `(expt -5 2.0)` возвращает `25.0`
- `(expt 8 -1)` возвращает `0` (оба аргумента целые)
- `(expt 4 -0.5)` возвращает `0.5`
- `(expt -1 -1.0)` возвращает `-1.0`

1.6.14. Функция *log*

Вычисляет натуральный логарифм.

```
(log <число>)
```

Тип аргумента: положительное целое или вещественное число.

Тип возвращаемого значения: вещественное число.

Примеры:

- `(log 1)` возвращает `0.0`
- `(log 3.8)` возвращает `1.335`

1.6.15. Функция *logand*

Вычисляет побитовое И для целых чисел. Полезна при работе с битовыми флагами (проверка признака замкнутости полилинии и т. п.).

```
(logand [<число1> [<число2> ... [<числоN>]...]])
```

Тип аргументов: целые числа.

Тип возвращаемого значения: целое число.

Побитовое И работает следующим образом. Каждое целое число может быть представлено в двоичном виде (иначе говоря, в виде суммы степеней числа 2, в которой перед степенью 2 стоит множитель 0 или 1; показатель степени в данной сумме может быть от 0 до 30). Например: $5 = 1 \times 2^0 + 0 \times 2^1 + 1 \times 2^2 = 101_2$, $3 = 1 \times 2^0 + 1 \times 2^1 = 11_2$.

31-й (старший) разряд в таком представлении отводится под знак (минусу соответствует 1, плюсу — 0). После перевода всех чисел в двоичное представление выполняется поразрядная (побитовая) операция И по следующему правилу:

```
(logand 0 0) = (logand 1 0) = (logand 0 1) = 0; (logand 1 1) = 1
```

Если аргументы не заданы, то возвращается `0`. Если задан только один аргумент `<число1>`, то возвращается его значение.

Примеры:

- `(logand 4 5)` возвращает `4`
- `(logand 12)` возвращает `12`
- `(logand)` возвращает `0`

1.6.16. Функция *logior*

Вычисляет побитовое ИЛИ для целых чисел.

```
(logior [<число1> [<число2> ... [<числоN>]...]])
```

Тип аргументов: положительные целые числа.

Тип возвращаемого значения: целое число.

Побитовое ИЛИ работает по следующему правилу:

```
(logior 1 0) = (logior 0 1) = (logior 1 1) = 1; (logior 0 0) = 0
```

Если аргументы не заданы, то возвращается `0`. Если задан только один аргумент, то возвращается `<число1>`.

Примеры:

- `(logior 4 5)` возвращает `5`
- `(logior 4 3)` возвращает `7`
- `(logior 5)` возвращает `5`

1.6.17. Функция *lsh*

Вычисляет результат побитового сдвига целого числа.

```
(lsh [<число> [<сдвиг>]])
```

Тип аргументов: целые числа.

Тип возвращаемого значения: целое число.

Побитовый сдвиг выполняется для двоичного представления чисел (о двоичном представлении *см. разд. 1.6.15*). Побитовый сдвиг выполняется влево, если `<сдвиг>` положителен, и вправо, если `<сдвиг>` отрицателен. Целые числа могут иметь не более чем 32 бита (нумерация битов от 0 для младшего разряда до 31 — этот разряд является знаковым), поэтому если старшие разряды при сдвиге влево выходят за 31-й разряд, то они теряются. При сдвиге вправо младшие разряды теряются.

Если аргументы не заданы, то возвращается `0`. Если задан только один аргумент или `<сдвиг>` равен 0, то возвращается `<число>`.

Примеры:

- `(lsh 2 1)` возвращает `4`
- `(lsh 4 1)` возвращает `8`
- `(lsh 3 29)` возвращает `1610612736`
- `(lsh 3 30)` возвращает `-1073741824`
- `(lsh 15 -1)` возвращает `7`
- `(lsh 15 -2)` возвращает `3`
- `(lsh 15 -3)` возвращает `1`
- `(lsh 15 -4)` возвращает `0`

1.6.18. Функция *boole*

Универсальная побитовая булева функция.

```
(boole <оператор> <число1> [<число2> ... ]])
```

Тип аргументов: целые числа.

Аргумент `<оператор>` должен иметь значение от 0 до 15, что соответствует 16-ти вариантам булевых функций. Остальные аргументы — произвольные целые числа.

Тип возвращаемого значения: целое число.

Номер любой булевой функции (они все двуместные, т. е. имеют два аргумента, каждый из которых может быть 0 или 1) кодируется следующим образом:

- если функция для пары `(1 1)` возвращает `1`, то к ее номеру добавляется $1 = 2^0$;
- если функция для `(1 0)` возвращает `1`, то добавляется $2 = 2^1$;
- если функция для `(0 1)` возвращает `1`, то добавляется $4 = 2^2$;
- если функция для `(0 0)` возвращает `1`, то добавляется $8 = 2^3$.

При таком кодировании знакомые нам функции `logand` и `logior` получат номера 1 и 7.

Булева функция применяется побитно к аргументам `<число1>` и `<число2>`, затем к результату и аргументу `<число3>` и т. д. Если заданы только два аргумента (`<оператор>` и `<число1>`), то возвращается значение аргумента `<число1>`.

Примеры:

- `(boole 1 4 5)` возвращает `4`
- `(boole 7 4 5)` возвращает `5`
- `(boole 15 3 -5)` возвращает `-1`
- `(boole 8 13 7)` возвращает `-16`

1.6.19. Функция *inters*

Геометрическая функция `inters` имеет следующий синтаксис:

```
(inters <точка1> <точка2> <точка3> <точка4> [<признак>])
```

Вычисляет координаты точки пересечения двух отрезков, один из которых ограничен точками, заданными в качестве аргументов `<точка1>` и `<точка2>`, а второй — точками, определенными аргументами `<точка3>` и `<точка4>`. Если заданный аргумент `<признак>` равен `nil`, тогда вместо точки пересечения отрезков ищется точка пересечения двух прямых, проходящих, соответственно, через первый и второй отрезки.

Типы аргументов: первые четыре аргумента — списки из двух или трех чисел, представляющих двумерные или трехмерные точки; `<признак>` — любое значение (важно только, равно оно `nil` или нет).

Возвращаемое значение: список с координатами точки пересечения или `nil`, если пересечения нет.

Если все точки трехмерные (т. е. аргументы заданы списками из трех чисел), то ищется пересечение отрезков или прямых в пространстве и возвращается, соответственно, список с координатами трехмерной точки. Если хотя бы одна точка двумерная, то вычисляется пересечение проекций отрезков или прямых на текущую плоскость построений и возвращается список с координатами двумерной точки.

Примеры:

- `(inters '(11.23 8.2) '(-9.1 2.2) '(12.7 8.59) '(-2.87 7.22) nil)` возвращает `(12.4883 8.57138)`
- `(inters '(11.23 8.2) '(-9.1 2.2) '(12.7 8.59) '(-2.87 7.22))` возвращает `nil`
- `(inters '(11.23 8.2 -2.0) '(-9.1 2.2 -2.0) '(12.7 8.59 -2.0) '(-2.87 7.22 -2.0) nil)` возвращает `(12.4883 8.57138 -2.0)`

1.6.20. Функция *polar*

Вычисляет координаты второй точки, отстоящей от первой точки, заданной аргументом <точка>, на расстояние, величина которого указана в качестве аргумента <расстояние>. При этом отрезок, построенный из первой точки во вторую, в проекции на текущую плоскость построений должен образовать с осью X угол, равный значению аргумента <угол> (в радианах).

```
(polar <точка> <угол> <расстояние>)
```

Аргументы: <точка> — список из двух или трех чисел, представляющих собой координаты двумерной или трехмерной точки; <угол> — вещественное число, представляющее собой угол, в радианах, с положительным направлением оси X текущей плоскости построений; <расстояние> — положительное вещественное число.

Примеры:

- `(polar '(3.1 9.3) 0.235 1.5)` возвращает `(4.55877 9.64926)`
- `(polar '(3.1 9.3 8.02) 0.235 1.5)` возвращает `(4.55877 9.64926 8.02)`

1.6.21. Функция *textbox*

Вычисляет две точки, расположенные по диагонали в вершинах прямоугольника, описанного вокруг примитива типа TEXT.

```
(textbox <список>)
```

Аргумент <список> должен представлять собой список с данными примитива типа TEXT (о структуре такого списка для объектов AutoCAD *см. разд. 1.12*). Если список неполный, то отсутствующие данные берутся по умолчанию. Минимальный объем такого списка должен содержать точечную пару с DXF-кодом 1 (т. е. сам текст надписи).

Пример:

- `(textbox '((1 . "Бермудский треугольник")))` возвращает `((0.0 -0.416667 0.0) (53.3333 2.5 0.0))` (такие данные, как точка начала текста, стиль, высота букв, взяты по умолчанию)

1.6.22. Функция *vports*

Выводит список с координатами видовых экранов в текущей вкладке (пространства модели или пространства листа).

```
(vports)
```

Возвращаемое значение — список с номерами и координатами видовых экранов. Форма списка зависит от значения системной переменной TILEMODE.

Если переменная TILEMODE равна 1, что соответствует вкладке **Model** (Модель), тогда координаты видовых экранов даются таким образом, что левый нижний угол графического экрана имеет координаты (0.0 0.0), а верхний правый — (1.0 1.0).

Примеры значений, возвращаемых при TILEMODE = 1:

- `((2 (0.0 0.0) (1.0 1.0)))` — для одноэкранной конфигурации;
- `((2 (0.0 0.5) (1.0 1.0)) (3 (0.0 0.0) (0.5 0.5)) (4 (0.5 0.0) (1.0 0.5)))` — для трехэкранной конфигурации (два экрана расположены снизу, один — сверху; экран с номером 2 является активным, рис. 1.3).

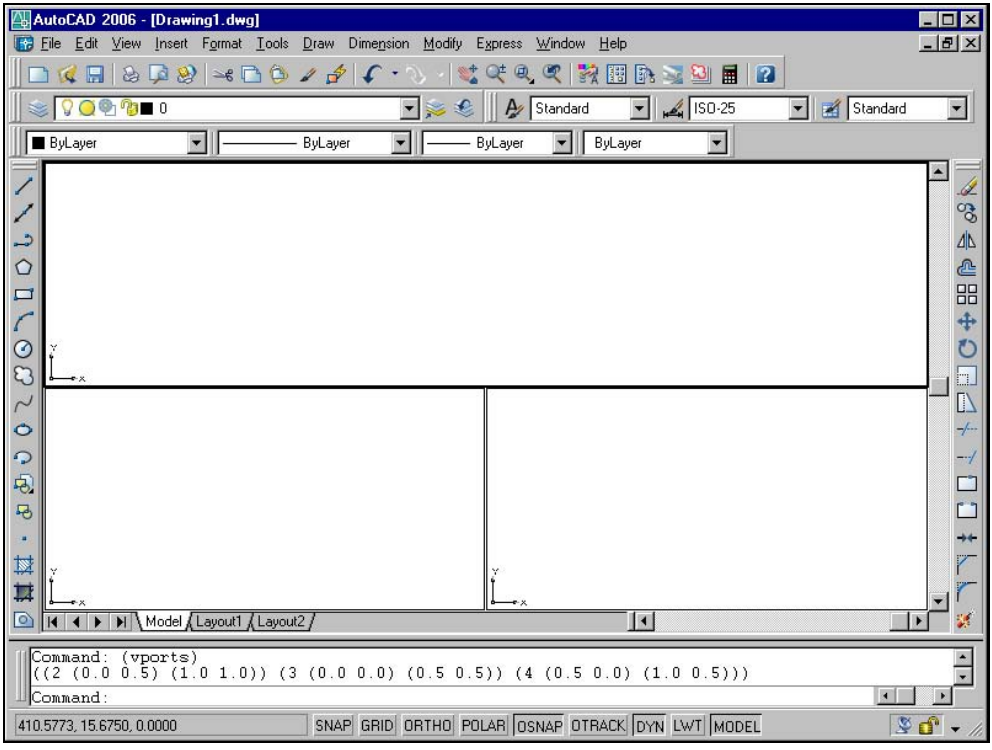


Рис. 1.3. Трехэкранная конфигурация видовых экранов в пространстве модели

Как правило, видовой экран с номером 1 зарезервирован для служебной информации.

Если системная переменная TILEMODE имеет значение 0, что соответствует одной из вкладок пространства листа, тогда координаты видовых экранов выводятся в системе координат листа.

Примеры значений, возвращаемых функцией `vports` при TILEMODE = 0:

- `((1 (0.0 0.0) (17.7288 9.0)))` — видовых экранов в листе нет;
- `((1 (0.0 0.0) (17.7288 9.0)) (2 (28.03 20.1613) (252.27 181.451)))` — для одноэкранной конфигурации (видовой экран неактивен);
- `((2 (28.03 20.1613) (252.27 181.451)) (1 (0.0 0.0) (17.7288 9.0)))` — для одноэкранной конфигурации (видовой экран активен);
- `((1 (0.0 0.0) (585.049 297.0)) (3 (123.125 69.5472) (278.111 179.409)) (2 (28.03 81.0797) (100.637 181.451)))` — для двухэкранной конфигурации (нет активных видовых экранов);
- `((3 (123.125 69.5472) (278.111 179.409)) (1 (0.0 0.0) (585.049 297.0)) (2 (28.03 81.0797) (100.637 181.451)))` — для двухэкранной конфигурации (активен экран справа);
- `((4 (17.3694 101.11) (141.966 186.086)) (1 (0.0 0.0) (17.7288 9.0)) (6 (141.966 16.1336) (266.563 101.11)) (5 (17.3694 16.1336) (141.966 101.11)) (3 (141.966 101.11) (266.563 186.086)))` — для четырехэкранной конфигурации (активен левый верхний видовой экран с номером 4, рис. 1.4).

В последних примерах видно, что видовой экран с номером 1 — это сам лист.

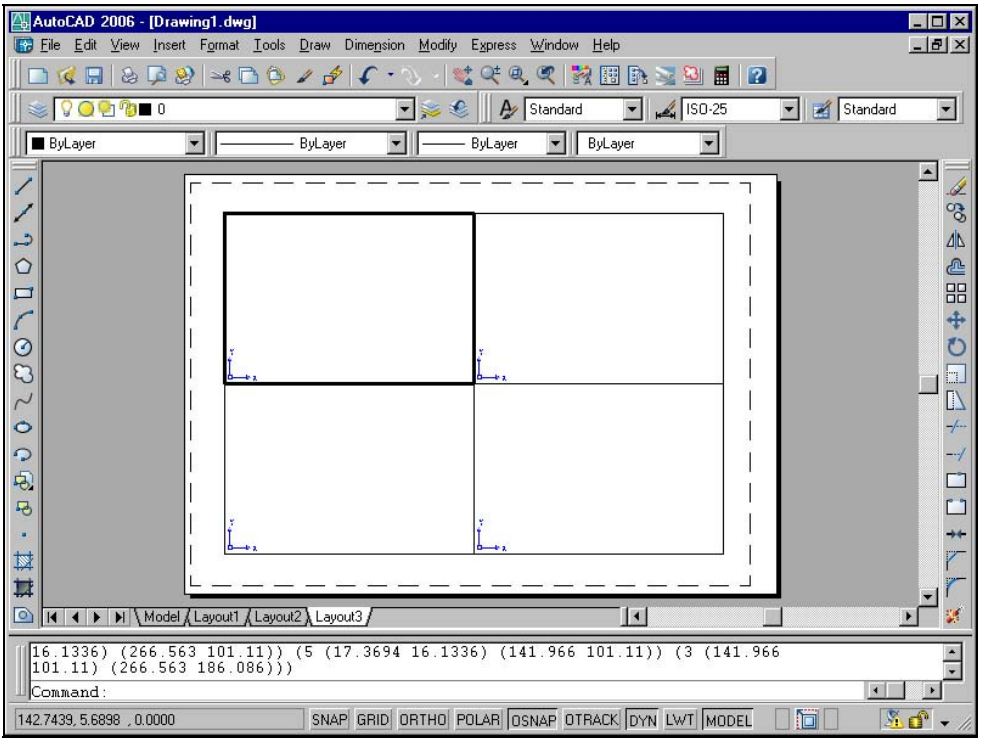


Рис. 1.4. Четырехэкранная конфигурация видовых экранов в пространстве листа

1.6.23. Функция *vl-symbol-value*

Возвращает значение символа, имя которого передано в качестве аргумента.

```
(vl-symbol-value '<символ>)
```

Аналогична операции получения значения символа с помощью восклицательного знака (`!`) в командной строке AutoCAD *(см. разд. 1.1.2)*.

Примеры:

- `(vl-symbol-value 'T)` возвращает `T`
- `(vl-symbol-value 'pi)` возвращает `3.14159`
- `(vl-symbol-value 'command)` возвращает `#<SUBR @09972ac8 COMMAND>`

1.6.24. Функция *set*

Присваивает значение символу, на который ссылается первый аргумент (сравните с функцией `setq` — *см. разд. 1.1.2*).

```
(set '<символ> <значение>)
```

Пример:

- `(set 'cir1 18.9)` возвращает `18.9`

Следующие выражения приводят к одному результату (присвоению нового значения символу `ry`):

- `(set 'ry 1)`
- `(setq ry 1)`
- `(set (read "ry") 1)`

1.7. Функции преобразования данных

В следующих разделах рассматриваются функции преобразования данных, а также функция `type`, определяющая тип конкретного символа. Функция `fix`, позволяющая усекать вещественные числа до ближайшего меньшего целого, приведена в *разд. 1.6.1*.

1.7.1. Функция *type*

Определяет тип символа.

```
(type <символ>)
```

Тип аргумента: символ (имя функции или переменной).

Если `<символ>` не является именем функции или переменной с присвоенным значением, то возвращается `nil`.

Другие варианты возвращаемого значения:

- `ENAME` — примитив AutoCAD;
- `FILE` — указатель (дескриптор) файла;
- `INT` — целое число;
- `LIST` — список, точечная пара или точечный список;
- `PAGETB` — таблица диспетчера страничной организации памяти;
- `PICKSET` — набор;
- `REAL` — вещественное число;
- `SAFEARRAY` — безопасный массив;
- `STR` — строка;
- `SUBR` — встроенная функция или функция, загруженная из VLX-файла (функция, определенная в ARX-приложении, имеет дополнительный признак `EXRXSUBR`);
- `SYM` — символ;
- `VARIANT` — вариант;
- `USUBR` — пользовательская функция, загруженная из LSP-файла;
- `VL-CATCH-ALL-APPLY-ERROR` — объект, создаваемый функцией `vl-catch-all-apply`;
- `VLA-OBJECT` — объект Visual LISP, использующий технологию ActiveX (указатель на структуру или экземпляр класса);
- `VLR-OBJECT` — реактор.

Некоторые из перечисленных вариантов возвращаемого значения доступны только при использовании других средств разработки приложений. Типы данных, доступные в LISP, приведены в *разд. 1.2.1*.

Примеры:

- `(type 73)` возвращает `INT`
- `(type 1-)` возвращает `SUBR`
- `(type "Рлс")` возвращает `STR`
- `(type '(1 81 264))` возвращает `LIST`

1.7.2. Функция *float*

Преобразует целое число в вещественное.

```
(float <число>)
```

Тип аргумента: целое или вещественное число.

Если <число> — целое, то возвращается результат его преобразования в вещественное. Если <число> — вещественное, то оно же и возвращается в качестве значения функции `float`.

Примеры:

- `(float 4)` возвращает `4.0`
- `(float -69)` возвращает `-69.0`
- `(float 4.3)` возвращает `4.3`

1.7.3. Функция *itoa*

Преобразует целое число в строку.

```
(itoa <целое>)
```

Тип аргумента: целое число.

Возвращается строка, являющаяся результатом преобразования аргумента <целое> в строковое представление.

Примеры:

- `(itoa 30)` возвращает `"30"`
- `(itoa -572)` возвращает `"-572"`
- `(itoa 0)` возвращает `"0"`

1.7.4. Функция *rtos*

Преобразует вещественное число в строку.

```
(rtos <число> [<режим> [<точность>]])
```

Аргументы: <число> — вещественное или целое число; <режим> — целое (номер режима представления вещественных чисел); <точность> — количество десятичных знаков в представлении вещественных чисел. Аргументы <режим> и <точность> имеют тот же смысл и принимают те же значения, что и системные переменные LUNITS и LUPREC. Если аргументы <режим> и <точность> опущены, то в качестве их значений принимаются текущие значения LUNITS и LUPREC. На форму возвращаемого значения может оказывать влияние системная переменная UNITMODE.

Допустимые значения аргумента <режим>:

- 1 — научный;
- 2 — десятичный;
- 3 — инженерный (футы и дюймы);

- 4 — архитектурный (футы и дюймы);
- 5 — дробный.

Возвращается строка, являющаяся результатом преобразования аргумента `<число>` в символьное представление с заданной точностью.

Примеры:

- `(rtos 10.9453 1 3)` возвращает `"1.095E+01"`
- `(rtos 10.9453 2 3)` возвращает `"10.945"`
- `(rtos 10.9453 3 3)` возвращает `"0'-10.945\""` (обратная косая черта предшествует двойной кавычке, как символу дюйма, чтобы не путать ее с двойной кавычкой, являющейся признаком конца строки)
- `(rtos 10.9453 4 3)` возвращает `"0'-11\""` (здесь тоже для вывода внутренней двойной кавычки использована обратная черта)
- `(rtos 10.9453 2 0)` возвращает `"11"`
- `(rtos 10.00000 2 3)` возвращает `"10"`, если значение системной переменной DIMZIN больше или равно 8 (замыкающие нули отбрасываются); возвращает `"10.000"` при других значениях DIMZIN
- `(rtos pi 2 11)` возвращает `"3.14159265359"`

Чаще всего применяется десятичный режим (номер 2).

1.7.5. Функция *atoi*

Преобразует строку в целое число.

```
(atoi <строка>)
```

Тип аргумента: строковый.

Возвращается целое число, для которого `<строка>` является строковым представлением.

Примеры:

- `(atoi "547")` возвращает `547`
- `(atoi "-26209")` возвращает `-26209`
- `(atoi "1234567890")` возвращает `1234567890`
- `(atoi "12345678901")` возвращает `2147483647` (в данном случае результат не соответствует действительности, поскольку число цифр в строковом представлении превысило допустимый максимум; следует воспользоваться функцией `atof`, *рассматриваемой ниже*)
- `(atoi "-15.28")` возвращает `-15` (точка и дробная часть отбрасывается)
- `(atoi "11d6")` возвращает `11` (недопустимые символы и следующие за ними теряются)

1.7.6. Функция *atof*

Преобразует строку в вещественное число.

```
(atof <строка>)
```

Тип аргумента: строковый.

Возвращается вещественное число, для которого `<строка>` является строковым представлением.

Примеры:

- `(atof "91")` возвращает `91.0`
- `(atof "-4.29")` возвращает `-4.29`
- `(atof "1234567890")` возвращает `1.23457e+009`
- `(atof "12345678901")` возвращает `1.23457e+010`

1.7.7. Функция *angtof*

Преобразует строку, представляющую значение угла в различных форматах, в вещественное число, являющееся величиной угла в радианах. Функция `angtof` по своему действию является обратной к функции `angtos`.

```
(angtof <строка> [<представление>])
```

Аргументы: `<строка>` — строка, `<представление>` — целое число. Значения аргумента `<представление>` соответствуют значениям системной переменной AUNITS. Если `<представление>` опущено, то в качестве его значения принимается текущее значение переменной AUNITS.

Допустимые значения аргумента `<представление>`:

- 0 — градусы;
- 1 — градусы, минуты, секунды;
- 2 — грады;
- 3 — радианы;
- 4 — топографические единицы.

Возвращается вещественное значение угла в радианах, для которого `<строка>` является строковым представлением (возможно, в другом формате представления углов). Если `<строка>` представлена в неправильном формате, то возвращается `nil`.

Примеры:

- `(angtof "22" 0)` возвращает `0.383972` (перевод угла из градусов в радианы)
- `(angtof "72.48g" 2)` возвращает `1.13851`

- `(angtof "7d13'20\"" 1)` возвращает `0.126052`
- `(angtof "_N 14d27'50\" _W" 4)` возвращает `1.82324`

1.7.8. Функция *angtos*

Преобразует значение угла в радианах в строку, в соответствии с форматом (функция `angtos` является обратной к функции `angtof`).

```
(angtos <угол> [<представление> [<точность>]])
```

Аргументы: `<угол>` — вещественное или целое значение угла в радианах; `<представление>` и `<точность>` — целые числа. Значения аргумента `<представление>` соответствуют значениям системной переменной AUNITS, значения аргумента `<точность>` — значениям системной переменной AUPREC. Если `<представление>` и `<точность>` опущены, то в качестве их значений принимаются текущие значения переменных AUNITS и AUPREC.

Возвращается `<строка>`, являющаяся строковым представлением значения угла. На форму результата оказывает влияние значение системной переменной UNITMODE. Необходимо также учитывать значение переменной ANGBASE.

Примеры:

- `(angtos 1.00 0)` возвращает `"57"`
- `(angtos 1 0)` возвращает `"57"`
- `(angtos 3.14000 0 3)` возвращает `"179.909"`
- `(angtos 3.14000 1 3)` возвращает `"179d54'31\""`
- `(angtos 3.14000 2 3)` возвращает `"199.899g"`
- `(angtos 3.14000 3 2)` возвращает `"3.14r"`
- `(angtos 1.294 4)` возвращает `"N 16d E"`
- `(angtos 1.294 4 4)` возвращает `"N 15d51'33\" E"`

1.7.9. Функция *distof*

Преобразует строку, представляющую вещественное значение в одном из форматов линейных единиц, в вещественное число. Функция `distof` является обратной по отношению к функции `rtos`.

```
(distof <строка> [<режим>])
```

Аргументы: `<строка>` — строка, представляющая число в одном из форматов; `<режим>` — целое число, значения которого соответствуют значениям системной переменной LUNITS *(см. разд. 1.7.4)*. Если аргумент `<режим>` опущен, то в качестве его значения принимается текущее значение системной переменной LUNITS.

Возвращаемое значение: вещественное число. Если <строка> имеет неправильную структуру, то возвращается nil.

Примеры:

- (distof "4" 2) возвращает 4.0
- (distof "32.53e+02" 1) возвращает 3253.0
- (distof "0'-10.4\"" 3) возвращает 10.4
- (distof "") возвращает nil

1.7.10. Функция *trans*

Преобразует координаты точки из одной системы координат (СК) в другую.

```
(trans <точка> <СК-из> <СК-в> [<признак>])
```

Аргументы: <точка> — точка (список из двух или трех чисел, соответствующий двумерной или трехмерной точке); <СК-из> — код СК, в которой подан аргумент <точка> (о допустимых значениях см. далее); <СК-в> — код СК, в которую нужно преобразовать точку. Необязательный аргумент <признак>, если он задан и отличен от nil (например, T), меняет интерпретацию аргумента <точка>: в этом случае список из двух или трех чисел рассматривается не как координаты точки, а как координаты вектора перемещения.

Аргументы <СК-из> и <СК-в> могут принимать следующие значения:

- имя примитива (в том виде, в каком оно возвращается функциями ssname, entsel, nentsel, entnext, entlast);
- трехмерный вектор выдавливания (вектор нормали к плоскости СК);
- целочисленный код от 0 до 3:
  - 0 — мировая СК (МСК);
  - 1 — текущая СК (ТСК, или ПСК);
  - 2 — экранная СК (ЭСК);
  - 3 — экранная СК листа (ЭСКЛ).

Имеются определенные условия использования этих кодов в аргументах <СК-из> и <СК-в> (иначе говоря, не всякое целочисленное значение <СК-из> сочетается с любым значением <СК-в> и наоборот).

Коды 0 и 1 могут использоваться в аргументах <СК-из> и <СК-в> в любых сочетаниях. Это означает, что если <СК-из> = 0, то <СК-в> может быть или любым целочисленным кодом (от 0 до 3), или именем примитива, или вектором выдавливания. Аналогично если <СК-в> = 0, то <СК-из> может быть или целочисленным кодом (от 0 до 3), или именем примитива, или вектором выдавливания.

Код 2 сочетается с разными вариантами, но его интерпретация зависит от цифрового кода другой СК. Если вторая СК имеет код 0 или 1, то первая СК (с кодом 2)

является ЭСК текущего видового экрана. Если вторая СК имеет код 3, то первая СК (с кодом 2) является ЭСК текущего видового экрана в пространстве модели.

Код 3 (ЭСКЛ) используется только в паре с кодом 2 другой СК (см. выше).

Имя примитива в качестве аргумента <СК-из> используется в том случае, когда координаты точки, хранящейся вместе с примитивом (например, начальная или конечная точки отрезка, центр окружности, вершина полилинии и т. п.), нужно пересчитать из системы координат объекта (СКО) в другую СК. В некоторых операциях (например, при работе с функцией entmake) возникает необходимость пересчитать точку в СКО — здесь имя примитива будет фигурировать в качестве аргумента <СК-в>.

Замечание

Некоторые объекты (например, трехмерные полилинии) не имеют особой СКО и хранят свои данные в МСК.

Свои преимущества имеет и способ, когда трехмерный вектор выдавливания применяется в качестве <СК-в> или <СК-из> (например, при проектировании объекта на плоскость с известным вектором нормали). Этот способ не может использоваться для тех объектов, у которых в качестве СКО фигурирует МСК.

Возвращаемым значением функции trans является список из трех координат точки (или вектора перемещения) в новой СК. Если значения <СК-из> и <СК-в> заданы противоречиво, то возвращается nil (например, если функция trans вызывается с кодом СК, равным 3, из пространства модели, а не из пространства листа).

Как было сказано выше, в качестве аргумента <точка> может фигурировать двумерная точка. В этом случае AutoCAD преобразует точку из двумерной в трехмерную, добавляя координату *Z* по следующему алгоритму.

Если задан аргумент <признак> и его значение отлично от nil (т. е. аргумент <точка> интерпретируется не как точка, а как вектор перемещения), то добавляемая координата *Z* получает значение 0. Если признак не задан или равен nil, то значение координаты *Z* зависит от аргумента <СК-из> и выбирается по табл. 1.1.

Таблица 1.1. *Выбор функцией* trans *координаты Z для двумерной точки*

СК-из	**Значение *Z***
МСК	0.0
ТСК	Текущий уровень (значение системной переменной ELEVATION)
СКО	0.0
ЭСК	В проекции на текущую плоскость построений (т. е. плоскость ТСК плюс текущий уровень)
ЭСКЛ	В проекции на текущую плоскость построений (т. е. плоскость ТСК плюс текущий уровень)

Рассмотрим пример. Предположим, в качестве ТСК выбрана система координат, у которой оси параллельны осям МСК, а начало находится в точке $X = 10$, $Y = 10$, $Z = 10$. Тогда:

- `(trans '(1 1 1) 0 1)` возвращает `(-9.0 -9.0 -9.0)`
- `(trans '(1 1 1) 0 1 T)` возвращает `(1.0 1.0 1.0)`

1.7.11. Функция *cvunit*

Преобразует значение из одной системы единиц измерения в другую.

```
(cvunit <значение> <е-из> <е-в>)
```

Аргументы: `<значение>` — число или список из двух или трех чисел; `<е-из>` — единица измерения, в которой задано `<значение>`; `<е-в>` — единица измерения, в которую надо перевести `<значение>`.

Возвращаемое значение имеет тот же тип (число или список), что и аргумент `<значение>`. Если преобразование невозможно, то возвращается `nil`.

Наименования единиц измерения, которые могут быть использованы в качестве аргументов `<е-из>` и `<е-в>`, должны быть перечислены в файле acad.unt текущего пользователя.

Примеры:

- `(cvunit 2 "minute" "second")` возвращает `120.0`
- `(cvunit 2 "минут" "секунд")` возвращает `120.0` (в русской версии)
- `(cvunit 1 "минута" "секунда")` возвращает `60.0` (в русской версии)
- `(cvunit 1 "minute" "hour")` возвращает `0.016667`
- `(cvunit '(10.0 2.0) "дюйм" "см")` возвращает `(25.4 5.08)`

1.7.12. Сообщения о неправильном типе

К часто встречающимся ошибкам относятся ошибки в типах аргументов, переданных функциям. Такие сообщения выводятся в командную строку AutoCAD, а при загруженной среде Visual LISP — в окно **Visual LISP Console** (Консоль Visual LISP). На рис. 1.5 приведен пример сообщения о том, что вместо целого числа в качестве аргумента функции `itoa` оказалась переданной строка `"2"`.

Все сообщения о неправильности типа аргумента выглядят таким образом: *bad argument type* (неверный тип аргумента) и далее идет сначала ключевое слово, обозначающее нужный тип (на рис. 1.5 — это `fixnump`), а затем — фактически встретившееся значение аргумента (`"2"`). В табл. 1.2 приведен список ключевых слов и соответствующих им типов данных (т. е. типу `fixnump` соответствует целое). Большинство ключевых слов похожи на функции LISP, проверяющие определенный тип данного.

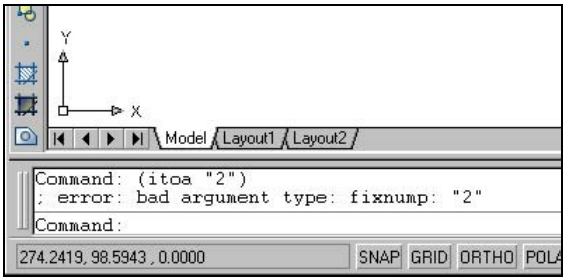


Рис. 1.5. Сообщение о неправильном типе

Таблица 1.2. *Соответствие между ключевыми словами и типами данных*

Ключевое слово	Требуемый тип данного
consp	Точечная пара
FILE	Дескриптор файла
fixnump	Целое
lentityp	Имя примитива
listp	Список
lselsetp	Набор
numberp	Число (целое или вещественное)
safearrayp	Безопасный массив
stringp	Строка
symbolp	Символ
Variantp	Вариант
VLA-OBJECT	VLA-объект
VL-Reactor	VLR-объект

1.8. Функции обработки строк

В следующих разделах разбираются функции, позволяющие выполнять операции над строками (постоянными или переменными). Рекурсивные функции П. В. Лоскутова для обработки строк рассмотрены в *главе 10*.

1.8.1. Функция *chr*

Получает буквенно-цифровой символ (строку из одного буквенно-цифрового символа) по коду действующей таблицы символов.

```
(chr <целое>)
```

Аргумент: целое число от 1 до 255. Если подается целое число вне указанного диапазона, то результат функции `chr` непредсказуем.

Тип возвращаемого значения: строковое.

Примеры:

- `(chr 192)` возвращает `"А"` (русская буква)
- `(chr 224)` возвращает `"а"` (русская буква)
- `(chr 32)` возвращает `" "` (пробел)
- `(chr 48)` возвращает `"0"`
- `(chr 50)` возвращает `"2"`

Замечание

Заглавные русские буквы (А—Я) имеют коды 192—223, а строчные (а—я) — коды 224—255.

1.8.2. Функция *ascii*

Получает код первого символа строки текста по действующей таблице символов.

```
(ascii <строка>)
```

Тип аргумента: строковый.

Тип возвращаемого значения: целое.

Примеры:

- `(ascii "А")` возвращает `192` (А — русская буква)
- `(ascii "A")` возвращает `65` (А — латинская буква)
- `(ascii "ABCD")` возвращает `65` (все буквенно-цифровые символы, кроме первого, игнорируются)
- `(ascii "")` возвращает `0`
- `(ascii "\"")` возвращает `34` (код двойной кавычки)

1.8.3. Функция *read*

Читает аргумент, являющийся строкой, до первого пробела (или до конца, если пробелов нет) и возвращает прочитанное значение как символ AutoLISP (т. е. имя функции или переменной).

```
(read <строка>)
```

Если обнаружено число, то оно преобразуется из строкового представления в обычное. Если в начале строки-аргумента указана двойная кавычка, то читается строка-аргумент до закрывающей двойной кавычки и возвращается прочитанная

между кавычками строка. Если в начале строки-аргумента находится открывающая круглая скобка, то читается строка-аргумент до закрывающей круглой скобки и возвращается прочитанный между скобками список.

Если строка-аргумент начинается с пробелов, то эти пробелы пропускаются до первого буквенно-цифрового символа, отличного от пробела. Если вся строка-аргумент состоит из пробелов, то возвращается `nil`.

Тип аргумента: строковый.

Тип возвращаемого значения: целое число, вещественное число, список или символ.

Примеры:

- `(read "A")` возвращает `A` (сравните с примером в *разд. 1.6.24*)
- `(read "    A  ")` возвращает `A`
- `(read "P Q R")` возвращает `P`
- `(read "17.2")` возвращает `17.2`
- `(read "150 entry")` возвращает `150`
- `(read "   ")` возвращает `nil`
- `(read "(11 10 9 8)")` возвращает `(11 10 9 8)`
- `(read "\"Таблица\" \"Формула\"")` возвращает `"Таблица"`

О некоторых нюансах функции `read` (не всегда понятных и приятных) можно прочитать в *приложении 4*.

1.8.4. Функция *strcase*

Преобразует строку в верхний или нижний регистр.

```
(strcase <строка> [<признак>])
```

Аргумент `<строка>` — текстовая строка, символы которой нужно преобразовать. Если аргумент `<признак>` опущен или равен `nil`, то функция `strcase` выполняет преобразование всех буквенно-цифровых символов строки в верхний регистр. Если `<признак>` задан и его значение отлично от `nil`, то выполняется преобразование строки в нижний регистр.

Символы, отличные от букв (например, цифры, знаки препинания), не преобразуются.

Примеры:

- `(strcase "Lower Corner  255")` возвращает `"LOWER CORNER  255"`
- `(strcase "Lower Corner  255" nil)` возвращает `"LOWER CORNER  255"`
- `(strcase "Lower Corner  255" T)` возвращает `"lower corner  255"`

- `(strcase "Lower Corner 255" 14)` возвращает `"lower corner 255"` (аргумент `<признак>` имеет значение `14`, что не равно `nil`, поэтому обрабатывается как `T`)

1.8.5. Функция *strcat*

Выполняет конкатенацию (сцепление) строк.

```
(strcat [<строка1> [<строка2> ... [<строкаN>] ... ]])
```

Тип аргументов: строковый. Если аргументы не заданы, то возвращается пустая строка (`""`).

Примеры:

- `(strcat "Happy N" "ew " "Year")` возвращает `"Happy New Year"`
- `(strcat (itoa 7) "-я линия")` возвращает `"7-я линия"`
- `(strcat "Петербург")` возвращает `"Петербург"`
- `(strcat)` возвращает `""`

1.8.6. Функция *strlen*

Вычисляет суммарную длину всех строк-аргументов (длина строки — это количество символов[2] в строке).

```
(strlen [<строка1> [<строка2> ... [<строкаN>] ... ]])
```

Тип аргументов — строковый. Если аргументы не заданы, то возвращается `0`.

Примеры:

- `(strlen "Happy N" "ew " "Year")` возвращает `14`
- `(strlen (itoa 7) "-я линия")` возвращает `9`
- `(strlen "Петербург")` возвращает `9`
- `(strlen)` возвращает `0`

1.8.7. Функция *substr*

Выделяет подстроку, начиная с заданной позиции.

```
(substr <строка> <начало> [<количество>])
```

Аргументы: `<строка>` — строка, из которой нужно выделить подстроку; `<начало>` — номер буквенно-цифрового символа в строке, с которого нужно начать выделение подстроки; `<количество>` — количество символов, которое нужно включить в подстроку. Аргумент `<начало>` — целое число (не меньше 1); номе-

[2] Один знак — один символ (кроме ограничивающих двойных кавычек).

ром 1 обозначается первый символ аргумента <строка>. Аргумент <количество> — целое число; если аргумент <количество> не указан или задан большим, чем осталось в анализируемой строке, начиная с позиции <начало>, то возвращается полный остаток аргумента <строка> от заданного места.

Если аргумент <количество> равен 0, то функция substr возвращает "".

Примеры:

- (substr "Happy New Year" 11 2) возвращает "Ye"
- (substr "Happy New Year" 7) возвращает "New Year"
- (substr "Happy New Year" 7 23) возвращает "New Year"
- (substr "Happy" 7 3) возвращает ""
- (substr "Happy" 3 0) возвращает ""

1.8.8. Функция *wcmatch*

Проверяет строку на соответствие ее символов образцу (шаблону).

(wcmatch <строка> <образец>)

Аргументы: <строка> — строка, структуру которой нужно проверить; <образец> — особая строка (образец), определяющая требования к проверяемой структуре. Аргумент <образец> может содержать как обычные алфавитно-цифровые знаки, так и специальные групповые знаки (wild-card-characters), рассматриваемые в табл. 1.3. Можно в строке <образец> задать сразу несколько шаблонов, на соответствие каждому из которых <строка> будет проверена; шаблоны должны разделяться с помощью запятой.

Таблица 1.3. *Специальные знаки, допустимые в образце*

Знак	Назначение
#	Заменяет одну любую цифру
@	Заменяет одну любую букву
. (точка)	Заменяет любой символ, не являющийся ни буквой, ни цифрой
*	Заменяет любую последовательность любых символов, даже пустую; может использоваться в любом месте аргумента <образец>
?	Заменяет ровно один любой символ
~	Если стоит в начале образца, заданного аргументом <образец>, то означает "все, кроме определенного аргументом <образец>"
[...]	Заменяет любой из символов, перечисленных в скобках
[~...]	Заменяет любой символ, кроме перечисленных в скобках
-	Между двумя однотипными знаками указывает на диапазон

Таблица 1.3 *(окончание)*

Знак	Назначение
, (запятая)	Разделяет два шаблона
` (обратный апостроф)	Превращает специальный знак в обычный

В квадратных скобках любой знак рассматривается не как специальный, поэтому его не надо предварять обратным апострофом. Символ \ является служебным, поэтому его использование отвечает правилам, изложенным в *разд. 1.2.1*.

Возвращаемое значение: T — если <строка> соответствует образцу, nil — если не соответствует.

Примеры:

- (wcmatch "Happy New Year" "H*") возвращает T, т. к. проверяемая строка начинается с буквы H
- (wcmatch "Happy New Year" "h*") возвращает nil (при проверке внутри строк символы H и h считаются разными)
- (wcmatch "Happy New Year" "??ppy*,*New*,* * *") возвращает T, т. к. все три проверяемых шаблона соблюдены
- (wcmatch "Happy New Year" "*#*") возвращает nil, т. к. нет ни одной цифры
- (wcmatch "Happy New Year" "H[adgt]*") возвращает T
- (wcmatch "Happy***day" "Happy`*`*`**") возвращает T
- (wcmatch "Year 2000" "@ea@ [1-6]000") возвращает T

1.8.9. Функция *vl-string->list*

Преобразует строку в список с кодами буквенно-цифровых знаков.

(vl-string->list <строка>)

Аргумент: текстовая строка.

Возвращаемое значение — список с кодами по действующей таблице символов или nil, если <строка> — пустая строка.

Примеры:

- (vl-string->list "LISP") возвращает (76 73 83 80)
- (vl-string->list "Москва") возвращает (204 238 241 234 226 224)
- (vl-string->list "") возвращает nil
- (vl-list->string (chr 0)) возвращает nil
- (vl-list->string (chr 256)) возвращает (0)

1.8.10. Функция *vl-list->string*

Преобразует список с кодами буквенно-цифровых знаков в строку.

```
(vl-list->string <список>)
```

Аргумент: список из целых неотрицательных чисел (каждое число должно быть не больше 256).

Возвращаемое значение — строка, в которой буквенно-цифровые символы имеют коды, взятые из аргумента `<список>`. Для пустого списка возвращается пустая строка (`""`).

Примеры:

- `(vl-list->string '(207 243 235 252 242 32 243 239 240 224 226 235 229 237 232 255))` возвращает `"Пульт управления"`
- `(vl-list->string '(207 243 235 252 242 32 0 0 243 239 240 224 226 235 229 237 232 255))` возвращает `"Пульт \000\000управления"`
- `(vl-list->string nil)` возвращает `""`

1.8.11. Функция *vl-string-elt*

Вычисляет код буквенно-цифрового символа, расположенного в строке на месте с заданным номером (первая буква имеет номер 0).

```
(vl-string-elt <строка> <позиция>)
```

Аргументы: `<строка>` — текстовая строка, `<позиция>` — целое число.

Возвращаемое значение — целое число (код буквенно-цифрового знака по действующей таблице символов).

Примеры:

- `(vl-string-elt "Маргарита" 5)` возвращает `240` (код буквы "р")
- `(vl-string-elt "Маргарита" 0)` возвращает `204` (код буквы "М")

1.8.12. Функция *vl-string-left-trim*

Убирает в начале строки все перечисленные символы.

```
(vl-string-left-trim <перечень> <строка>)
```

Аргументы: текстовые строки.

Возвращаемое значение — значение аргумента `<строка>`, из которого слева убраны все символы, перечисленные в аргументе `<перечень>`.

Функция удаляет все требуемые буквенно-цифровые знаки, начиная с первого символа строки, до тех пор, пока не встретится символ, который не входит в перечень удаляемых.

Примеры:

- `(vl-string-left-trim "корпус" "проспект")` возвращает `"ект"`
- `(vl-string-left-trim " " "  узел")` возвращает `"узел"`

1.8.13. Функция *vl-string-right-trim*

Убирает в конце строки все перечисленные символы.

```
(vl-string-right-trim <перечень> <строка>)
```

Аргументы: текстовые строки.

Возвращаемое значение — значение аргумента `<строка>`, из которого справа убраны все символы, перечисленные в аргументе `<перечень>`.

Функция удаляет все требуемые буквенно-цифровые знаки, начиная с последнего символа строки, до тех пор, пока не встретится символ, который не входит в перечень удаляемых.

Примеры:

- `(vl-string-right-trim "лоция" "дистанция")` возвращает `"дистан"`
- `(vl-string-right-trim " Mary" "commentary  ")` возвращает `"comment"`

1.8.14. Функция *vl-string-trim*

Убирает в начале и в конце строки все перечисленные символы.

```
(vl-string-trim <перечень> <строка>)
```

Аргументы: текстовые строки.

Возвращаемое значение — значение аргумента `<строка>`, из которого слева и справа убраны все символы, перечисленные в аргументе `<перечень>`.

Функция является комбинацией функций `vl-string-left-trim` и `vl-string-right-trim`.

Примеры:

- `(vl-string-trim "декларация" "дистанция")` возвращает `"стан"`
- `(vl-string-trim " " "    comment  ")` возвращает `"comment"`

1.8.15. Функция *vl-string-mismatch*

Сверяет две строки, начиная с заданных позиций, и возвращает количество совпавших последовательностей буквенно-цифровых символов.

```
(vl-string-mismatch <строка1> <строка2> [<позиция1> [<позиция2> [<регистр>]]])
```

Аргументы: `<строка1>` и `<строка2>` — строки, `<позиция1>` и `<позиция2>` — числа (номера позиций в сверяемых строках; отсчет позиций в строке начинается с 0);

<регистр> — любое значение (имеет значение только nil или не nil). По умолчанию аргументы <позиция1> и <позиция2> принимают значение 0, <регистр> — nil. Если аргумент <регистр> задан и имеет значение, отличное от nil, то при сверке регистр символов игнорируется.

Примеры:

- (vl-string-mismatch "декларация" "дистанция") возвращает 1
- (vl-string-mismatch "декларация" "ДИСТАНЦИЯ" 7 6) возвращает 0
- (vl-string-mismatch "декларация" "ДИСТАНЦИЯ" 7 6 T) возвращает 3
- (vl-string-mismatch "декларация" "ДИСТАНЦИЯ" 7 6 44) возвращает 3
- (vl-string-mismatch "декларация" "ДИСТАНЦИЯ" 7 6 nil) возвращает 0
- (vl-string-mismatch "Merry Christmas" "MER" 0 0 T) возвращает 3

1.8.16. Функция *vl-string-position*

Ищет буквенно-цифровой знак с определенным кодом в строке, начиная с заданной позиции (возможен поиск с конца строки).

```
(vl-string-position <код> <строка> [<начало> [<обратно>]])
```

Аргументы: <код> — код (положительное целое число), с которым разыскивается буквенно-цифровой символ; <строка> — анализируемая строка; <начало> — номер позиции, начиная с которого анализируется строка (первый символ строки имеет номер 0); <обратно> — признак поиска с конца строки (если не nil, то поиск идет с конца).

Возвращаемое значение — число, являющееся номером позиции первого найденного в аргументе <строка> символа с заданным кодом. Если требуемый символ не обнаружен, то возвращается nil.

Примеры:

- (vl-string-position 49 "1111" 0) возвращает 0
- (vl-string-position 49 "1111" 1) возвращает 1
- (vl-string-position 49 "1111" 1 T) возвращает 3
- (vl-string-position 53 "1111") возвращает nil
- (vl-string-position (ascii "o") "Moscow") возвращает 1
- (vl-string-position (ascii "o") "Moscow" 0 T) возвращает 4

1.8.17. Функция *vl-string-search*

Ищет заданную последовательность буквенно-цифровых символов (образец) в строке, начиная с заданной позиции.

```
(vl-string-search <образец> <строка> [<начало>])
```

Аргументы: <образец> и <строка> — текстовые строки; <начало> — номер позиции, с которой начинается анализ аргумента <строка> (по умолчанию — 0). Отсчет позиций в строке начинается с 0.

Возвращаемое значение — целое число с номером позиции, начиная с которой обнаружено присутствие аргумента <образец> в аргументе <строка>. Если <образец> не обнаружен, то возвращается `nil`.

Примеры:

- `(vl-string-search "in" "Training")` возвращает `3`
- `(vl-string-search "IN" "Training")` возвращает `nil`
- `(vl-string-search "in" "Training" 4)` возвращает `5`
- `(vl-string-search "in" "Training" 6)` возвращает `nil`

1.8.18. Функция *vl-string-subst*

Заменяет в строке одну последовательность буквенно-цифровых символов на другую, начиная с определенного номера позиции.

```
(vl-string-subst <новые> <старые> <строка> [<начало>])
```

Аргументы: <новые>, <старые> и <строка> — текстовые строки; <начало> — номер позиции в аргументе <строка>, с которого начинается поиск нужной последовательности символов для замены (по умолчанию — 0). Первый символ в строке имеет номер 0.

Возвращаемое значение — строка, являющаяся результатом выполнения замены последовательности символов в аргументе <строка>. Если последовательность символов <старые> встречается в аргументе <строка> несколько раз, то производится только первая замена.

Примеры:

- `(vl-string-subst "цып" "ва" "валенок")` возвращает `"цыпленок"`
- `(vl-string-subst "12" "a" "a para" 0)` возвращает `"12 para"`
- `(vl-string-subst "12" "a" "a para" 2)` возвращает `"a p12ra"`
- `(vl-string-subst "12" "a" "a para" 5)` возвращает `"a par12"`
- `(vl-string-subst "12" "b" "a para" 0)` возвращает `"a para"`

1.8.19. Функция *vl-string-translate*

Заменяет в строке символы одной группы на символы другой группы.

```
(vl-string-translate <группа1> <группа2> <строка>)
```

Аргументы: текстовые строки.

Возвращаемое значение — строка с замененными символами (если символы из аргумента <группа1> найдены в аргументе <строка>). Обычно количество символов в аргументе <группа1> равно количеству символов в аргументе <группа2>. Если количество символов в первом и втором аргументах отличаются, то в каждой из групп используются только первые символы (лишние, не имеющие пары в другой группе, игнорируются).

Примеры:

- (vl-string-translate "19" "ab" "part 1921") возвращает "part ab2a"
- (vl-string-translate "abc" "w" "kabca") возвращает "kwbcw"

1.8.20. Функция *vl-symbol-name*

Выводит имя символа в форме текстовой строки.

```
(vl-symbol-name '<символ>)
```

Аргумент — символ LISP.

Возвращаемое значение — строка.

Примеры:

- (vl-symbol-name 'pi) возвращает "pi"
- (vl-symbol-name 'append) возвращает "append"

1.9. Функции обработки списков и точечных пар

В этом разделе рассматриваются функции, работающие со списками и точечными парами. *Точечная пара* — это особый двухэлементный список вида (a . b), в котором в качестве разделителя выступает точка. Первый элемент такого списка называют *DXF-кодом*, а второй — *данными* этого кода. Основной способ создания точечных пар — функция cons. Точечные пары используются для работы с примитивами рисунка *(см. разд. 1.12)*.

Есть отличия в работе некоторых функций со списками и с точечными парами.

1.9.1. Функция *list*

Объединяет элементы в список. Функция list — это основная функция, позволяющая создать список. Обращение к функции:

```
(list [<элемент1> [<элемент2> ... [<элементN>] ... ]])
```

В качестве аргументов, образующих список, могут выступать любые объекты, которыми оперирует LISP. Самый распространенный список — это список из двух или трех вещественных чисел, который представляет точку.

В качестве элементов списка могут выступать другие списки или точечные пары.

Примеры:

- `(list 3.62 1.49 4.12)` возвращает `(3.62 1.49 4.12)`
- `(list 2.3 nil T)` возвращает `(2.3 nil T)`
- `(list 1 "crt" (list 0 1 2))` возвращает `(1 "crt" (0 1 2))` (третьим элементом нового списка является список из чисел 0, 1 и 2)
- `(list 1 '(0 . "LINE"))` возвращает `(1 (0 . "LINE"))` (вторым элементом нового списка является точечная пара `(0 . "LINE")`)

1.9.2. Функция *append*

Выполняет слияние списков в один.

```
(append [<список1> [<список2> ... [<списокN>] ... ]])
```

Все аргументы должны быть списками. Точечные пары в качестве аргументов не допускаются (в то же время точечные пары могут являться элементами объединяемых списков). Возвращаемое значение — список, в котором после элементов списка, указанного в качестве первого аргумента, добавлены элементы второго и последующего списков.

Если присутствует только один аргумент, то его значение и возвращается в качестве значения функции `append`. Если аргументы опущены, то возвращается `nil`.

Примеры:

- `(append '(8 33 -6) (list -10 0 22))` возвращает `(8 33 -6 -10 0 22)`
- `(append '(200 -6) '(nil T nil))` возвращает `(200 -6 nil T nil)`
- `(append '(9 155 -26))` возвращает `(9 155 -26)`
- `(append)` возвращает `nil`
- `(append nil '(1))` возвращает `(1)` (`nil` рассматривается как список нулевой длины, т. е. `'()`)
- `(append '(1 69 -3) '((62 . 8)))` возвращает `(1 69 -3 (62 . 8))` (в этом примере второй аргумент — это список, в котором единственным элементом является точечная пара `(62 . 8)`)

1.9.3. Функция *nth*

Извлекает из списка элемент по порядковому номеру (нумерация элементов списка выполняется слева направо и начинается с нуля).

```
(nth <номер> <список>)
```

Типы аргументов: <номер> — целое неотрицательное число, <список> — непустой список любой длины.

Возвращается значение элемента на соответствующем месте списка. Если номер превышает длину списка или равен ей, то возвращается `nil`.

Примеры:

- `(nth 0 '(11 12 13 14))` возвращает `11`
- `(nth 1 '(11 12 13 14))` возвращает `12`
- `(nth 2 '(11 12 13 14))` возвращает `13`
- `(nth 3 '(11 12 13 14))` возвращает `14`
- `(nth 4 '(11 12 13 14))` возвращает `nil`

1.9.4. Функция *reverse*

Переворачивает список (образует новый список, в котором элементы переставлены в порядке, противоположном порядку элементов исходного списка).

```
(reverse <список>)
```

Тип аргумента: <список> — любой список (точечные пары не допускаются) или `nil`.

Возвращаемое значение — перевернутый список или `nil`.

Примеры:

- `(reverse '(11 12 13 14 15))` возвращает `(15 14 13 12 11)`
- `(reverse '(51))` возвращает `(51)`
- `(reverse nil)` возвращает `nil`
- `(reverse '())` возвращает `nil`

1.9.5. Функция *car*

Извлекает первый элемент списка, точечной пары или точечного списка.

```
(car <список>)
```

Аргументом функции могут быть список, точечная пара, точечный список или `nil`.

Если аргументом является список или точечный список, то возвращается первый элемент списка; если аргументом является точечная пара, то возвращается DXF-код (первый элемент точечной пары). Если в качестве аргумента передан `nil` (или эквивалентный ему пустой список), то возвращается `nil`.

Примеры:

- `(car '(11 12 13 14 15))` возвращает `11`
- `(car '(0 . "LINE"))` возвращает `0`
- `(car '(2 4 . 11))` возвращает `2`
- `(car '())` возвращает `nil`

1.9.6. Функция *cdr*

Извлекает подсписок без первого элемента или данные точечной пары.

```
(cdr <список>)
```

Аргументом функции могут быть список, точечная пара, точечный список или `nil`.

Если аргументом является список или точечный список, то возвращается такой же список, но без первого элемента; если аргументом является точечная пара, то возвращаются данные (второй элемент точечной пары). Если в качестве аргумента передан `nil` (или эквивалентный ему пустой список), то возвращается `nil`.

Примеры:

- `(cdr '(11 12 13 14))` возвращает `(12 13 14)`
- `(cdr '(0 . "LINE"))` возвращает `"LINE"`
- `(cdr '(2 4 . 11))` возвращает `(4 . 11)`
- `(cdr '())` возвращает `nil`

1.9.7. Функции *caar*, *cadr* и аналогичные им

От функций `car` и `cdr` путем их повторения в разных комбинациях от двух до четырех раз в языке AutoLISP образованы следующие функции (всего их двадцать восемь):

- `(caar <список>)` — равносильно `(car (car <список>))`;
- `(cadr <список>)` — равносильно `(car (cdr <список>))`;
- `(cdar <список>)` — равносильно `(cdr (car <список>))`;
- `(cddr <список>)` — равносильно `(cdr (cdr <список>))`;
- `(caaar <список>)` — равносильно `(car (car (car <список>)))`;
- `(caadr <список>)` — равносильно `(car (car (cdr <список>)))`;
- `(cadar <список>)` — равносильно `(car (cdr (car <список>)))`;
- `(caddr <список>)` — равносильно `(car (cdr (cdr <список>)))`;
- `(cdaar <список>)` — равносильно `(cdr (car (car <список>)))`;

- (cdadr <список>) — равносильно (cdr (car (cdr <список>)));
- (cddar <список>) — равносильно (cdr (cdr (car <список>)));
- (cdddr <список>) — равносильно (cdr (cdr (cdr <список>)));
- (caaaar <список>) — равносильно (car (car (car (car <список>))));
- (caaadr <список>) — равносильно (car (car (car (cdr <список>)))).

Аналогичным методом раскрываются оставшиеся функции (caadar, caaddr, cadaar, cadadr, caddar, cadddr, cdaaar, cdaadr, cdadar, cdaddr, cddaar, cdddar, cddadr, cddddr).

Примеры:

- (cadr '(11 12 13 14)) возвращает 12
- (cddr '(11 12 13 14)) возвращает (13 14)
- (cdddr '(11 12 13 14)) возвращает (14)
- (cadddr '(11 12 13 14)) возвращает 14
- (cddddr '(11 12 13 14)) возвращает nil
- (cadr '((11 12 13 14))) возвращает nil
- (caar '((11 12 13 14))) возвращает 11
- (cadr '((11 12 13 14) (21 22 23 24))) возвращает (21 22 23 24)
- (cddr '((11 12 13 14) (21 22 23 24))) возвращает nil
- (cdadr '((11 12 13 14) (21 22 23 24))) возвращает (22 23 24)
- (cadadr '((11 12 13 14) (21 22 23 24))) возвращает 22
- (cddadr '((11 12 13 14) (21 22 23 24))) возвращает (23 24)

1.9.8. Функция *cons*

Добавляет к списку первый элемент или создает точечную пару.

(cons <аргумент1> <аргумент2>)

Если <аргумент2> является списком, то функция cons добавляет в него <аргумент1> в качестве нового первого элемента.

Если <аргумент2> является атомом (т. е. не списком), то функция cons создает точечную пару типа (<аргумент1> . <аргумент2>). Функция cons является основным инструментом создания точечных пар, которые совместно с целочисленными первыми элементами используются в структуре примитивов AutoCAD *(см. разд. 1.12)*.

Примеры:

- (cons 5.4 '(8.3 -1.0)) возвращает (5.4 8.3 -1.0)
- (cons "Happy" '("New" "Year")) возвращает ("Happy" "New" "Year")

- `(cons 5 1)` возвращает `(5 . 1)`
- `(cons 8 "0")` возвращает `(8 . "0")`
- `(cons 62 5)` возвращает `(62 . 5)`
- `(cons 47 nil)` возвращает `(47)` (`nil` интерпретируется как пустой список, т. е. `'()`)

1.9.9. Функция *vl-list**

Создает список, точечную пару, точечный список или возвращает значение первого аргумента, если остальные аргументы не заданы.

```
(vl-list* <объект1> [<объект2> ... [<объектN>]...])
```

Аргументы — любые.

Если задан только один аргумент, то возвращается его значение.

Если количество аргументов равно двум, а `<объект2>` — атом (т. е. не список), то возвращается точечная пара из значений двух аргументов. Если количество аргументов равно двум, а `<объект2>` — список, то возвращается список, в котором первым элементом является значение аргумента `<объект1>`, а остальными элементами — элементы списка `<объект2>` (т. е. фактически вместо функции `vl-list*` выполняется функция `cons`).

Если количество аргументов больше двух, а `<объектN>` — атом, то возвращается *точечный список* (см. пример ниже), в котором значения первых аргументов отделены точкой от значения последнего аргумента. Если количество аргументов больше двух, а `<объектN>` — список, то возвращается список, в котором на первых местах стоят значения всех аргументов, кроме последнего аргумента, а в конце добавлены элементы списка `<объектN>`.

Примеры:

- `(vl-list* 63)` возвращает `63`
- `(vl-list* 68 23)` возвращает `(68 . 23)`
- `(vl-list* 41 '(55))` возвращает `(41 55)`
- `(vl-list* 22 '(6 0))` возвращает `(22 6 0)`
- `(vl-list* 11 83 9)` возвращает `(11 83 . 9)` (это и есть точечный список)
- `(vl-list* 60 2 59 '(7 10))` возвращает `(60 2 59 7 10)`

1.9.10. Функция *member*

Проверяет принадлежность элемента списку.

```
(member <элемент> <список>)
```

Типы аргументов: `<элемент>` — любой, `<список>` — список.

Если функция member обнаруживает <элемент> в аргументе <список>, то возвращает остаток списка, начиная с этого места; если <элемент> в аргументе <список> не обнаруживается, то функция возвращает nil.

Примеры:

- (member 1 '(3 5 1 2)) возвращает (1 2)
- (member 4 '(4 290 1 4)) возвращает (4 290 1 4)
- (member 7 '(4 290 1 4)) возвращает nil
- (member 12 nil) возвращает nil

1.9.11. Функция *assoc*

Функция assoc применяется к сложному (ассоциированному) списку, в котором элементами являются списки или точечные пары, и выбирает из этих элементов (внутренних списков и точечных пар) тот, у которого первый элемент имеет заданное значение.

```
(assoc <код> <список>)
```

Если в аргументе <список> имеется несколько элементов, удовлетворяющих требуемому условию, то в качестве возвращаемого значения выбирается первый из них. Функция assoc — основной инструмент в операции, когда из списка с данными примитива AutoCAD нужно выбрать тот элемент, который содержит точечную пару с нужным DXF-кодом свойства (цвета, слоя, типа линий, веса и т. д.).

Возвращаемое значение — точечная пара или список, являющиеся элементами аргумента <список> и имеющие в качестве своего первого элемента заданный <код>.

Примеры:

- (assoc 8 '((8 . "Walls") (6 . "DASHED"))) возвращает (8 . "Walls")
- (assoc 7 '((8 . "Walls") (6 . "DASHED"))) возвращает nil
- (assoc 62 '((62 . 4) (6 . "DASHED"))) возвращает (62 . 4)
- (assoc 10 '((10 51.2 6.5 2.7) (10 122.01 72.94 80.39))) возвращает (10 51.2 6.5 2.7)

1.9.12. Функция *apply*

Применяет функцию, имя которой задано в первом аргументе после апострофа, поочередно ко всем элементам списка, заданного аргументом <список>.

```
(apply '<функция> <список>)
```

Примеры:

- `(apply '* (list 2 4 5))` равносильно `(* 2 4 5)`, поэтому возвращает `40`
- `(apply 'min (list 12 7 20))` возвращает `7`
- `(apply 'command (list "_.CIRCLE" "75,64" 120.0))` выполняет команду построения окружности по центру и радиусу и возвращает `nil`

Последний пример показывает, что функция `apply` удобна, когда определенную операцию нужно выполнить над аргументами, количество которых заранее не известно. Следует сформировать список, к которому затем нужно применить функцию `apply` с именем требуемой функции в качестве первого аргумента. Читателю следует обратить внимание на употребление апострофа перед аргументом.

1.9.13. Функция *mapcar*

Применяет функцию, имя которой задано аргументом `<функция>`, сначала к первым элементам каждого из последующих аргументов, являющихся списками (`<список1>`, …, `<списокN>`), затем — ко вторым элементам и т. д.; результаты объединяются в новый список, который является возвращаемым значением.

```
(mapcar '<функция> <список1> [<список2> ... [<списокN>] ... ])
```

Примеры:

- `(mapcar '* '(0 4) '(8 3))` возвращает `(0 12)`, т. к. первый элемент результирующего списка вычисляется как `(* 0 8)`, а второй — как `(* 4 3)`
- `(mapcar '- '(9 3) '(4 2))` возвращает `(5 1)`, т. к. первый элемент результирующего списка вычисляется как `(- 9 4)`, а второй — как `(- 3 2)`

1.9.14. Функция *vl-every*

Проверяет, возвращает ли тест-функция результат `T` со всеми списками-аргументами (функция применяется сначала к первым элементам всех списков, затем ко вторым элементам и т. д.).

```
(vl-every '<тест-функция> <список1> [<список2> ... [<списокN>]...])
```

Аргументы: `<тест-функция>` — любая функция (тест-функция), которая может выполнить свою проверку с тем же количеством аргументов, что и количество списков-аргументов функции `vl-every`, и которая возвращает `T` или `nil`; `<список1>`, ..., `<списокN>` — списки, из которых передаются тест-функции сначала первые элементы, затем — вторые элементы и т. д. до исчерпания хотя бы одного из списков-аргументов.

В качестве тест-функций могут выступать как обычные функции, так и функции, созданные с помощью `lambda` (lambda-функции). Для указания имени тест-

функции можно применять специальную функцию `function` *(см. разд. 1.16)*. При использовании `function` апостроф не ставится.

Если для элементов с одинаковыми номерами (первых, вторых и т. д.) всех списков тест-функция возвращает `T`, то и функция `vl-every` возвращает `T`. В других случаях функция `vl-every` возвращает `nil`.

Примеры:

- `(vl-every '= '(15 3) '(15.0 3.0 9.0))` возвращает `T` (поскольку в первом списке только два элемента, то проверка на равенство выполняется два раза, и оба раза результат — `T`)
- `(vl-every (function =) '(15 3) '(15.0 3.0 9.0))` возвращает `T` (аналогично предыдущему)
- `(vl-every '< '(4 4) '(15 7) '(15.0 9.9))` возвращает `nil` (для первых элементов функция `<` дает результат `nil`)
- `(vl-every '< nil '(4 4) '(15.0 9.9))` возвращает `T` (`nil` — это список из нуля элементов, поэтому проверка выполняется 0 раз и результат должен быть `T`)

1.9.15. Функция *foreach*

Функция `foreach` позволяет организовать циклическое выполнение выражений (их может быть несколько), поочередно подставляя вместо переменной цикла (аргумента `<имя>`) элементы списка.

```
(foreach <имя> <список> [<выражение1> [<выражение2>... [<выражениеN>]...]])
```

Аргументы: `<имя>` — символ, играющий роль временной переменной цикла; `<список>` — список, элементы которого поочередно присваиваются в качестве значения переменной `<имя>`; `<выражение1>`, ..., `<выражениеN>` — выполняемые выражения.

Функция `foreach` возвращает значение последнего вычисленного выражения. Если аргументы `<выражение1>`, …, `<выражениеN>` не заданы, то функция возвращает `nil`.

Пример:

- `(setq s1 0 s2 1)`
- `(foreach p '(1 2 3 4 5 6) (setq s1 (+ s1 p)) (setq s2 (* s2 p)))` возвращает `720` (результат вычисления `s2` — произведения чисел списка); кроме того, в переменной `s1` сформируется сумма чисел списка (21).

1.9.16. Функция *eval*

Функция `eval` позволяет вычислить (evaluate) аргумент как выражение. Если аргумент является списком, то в нем на первом месте должно стоять имя функции.

```
(eval <выражение>)
```

Пример:

- `(setq mylist (list 'max 2.4 3.1 1.47))`

  `(eval mylist)` возвращает `3.1` (вычисляется выражение `(max 2.4 3.1 1.47)`)

1.9.17. Функция *quote*

Функция `quote` является обратной по отношению к функции `eval` и позволяет обрабатывать выражение, не вычисляя его. Сокращенным вариантом имени функции `quote` является апостроф.

```
(quote <элементы выражения>)
```

Примеры:

- `(quote 6 7 8 2 1)` возвращает `(6 7 8 2 1)`
- `'(16 23 90)` возвращает `(16 23 90)`
- `'(0 . "ARC")` возвращает `(0 . "ARC")`

Апостроф очень часто используется при построении постоянных списков и точечных пар. См. также функцию `function` *(разд. 1.16)*.

1.9.18. Функция *acad_strlsort*

Позволяет отсортировать по возрастанию список, элементами которого являются строки.

```
(acad_strlsort <список>)
```

Примеры:

- `(acad_strlsort '("Пн" "Вт" "Ср" "Чт" "Пт" "Сб" "Вс"))` возвращает `("Вс" "Вт" "Пн" "Пт" "Сб" "Ср" "Чт")`
- `(acad_strlsort '("Июнь" "Июль" "Август"))` возвращает `("Август" "Июль" "Июнь")`

1.9.19. Функция *last*

Функция `last` возвращает последний элемент списка, заданного аргументом `<список>`, или `nil`, если список пуст.

```
(last <список>)
```

Примеры:

- `(last '())` возвращает `nil`
- `(last '(54 12 20))` возвращает `20`
- `(last (list (11 23) (12 20)))` возвращает `(12 20)`

1.9.20. Функция *subst*

Заменяет в списке старый элемент (все вхождения) на новый.

```
(subst <новый> <старый> <список>)
```

Аргументы: `<новый>` и `<старый>` — любые символы LISP; `<список>` — это список, в котором элемент `<старый>` нужно заменить на элемент `<новый>`.

Возвращаемое значение — список с замененным, возможно, элементом.

Примеры:

- `(subst 7 2 '(22 3 2 0))` возвращает `(22 3 7 0)`
- `(subst 7 2 '(3 4 5 1))` возвращает `(3 4 5 1)`
- `(subst 7 2 '(2 2 2 0))` возвращает `(7 7 7 0)`
- `(subst "5" 11 '(11 5 17 4))` возвращает `("5" 5 17 4)`

1.9.21. Функция *length*

Возвращает длину (количество элементов) списка.

```
(length <список>)
```

Аргумент — список (может быть `nil`). Точечные пары и точечные списки не допускаются (сравните с функцией `vl-list-length`).

Возвращаемое значение — неотрицательное целое число.

Примеры:

- `(length '())` возвращает `0`
- `(length nil)` возвращает `0`
- `(length '(4 2 1 100 4))` возвращает `5`
- `(length '((4 2 1 100 4)))` возвращает `1`

1.9.22. Функция *vl-list-length*

Возвращает длину (количество элементов) списка.

```
(vl-list-length <список>)
```

Аргумент — список, точечная пара или точечный список (см. описание функции `vl-list*` в *разд. 1.9.9*), может быть `nil`.

Возвращаемое значение: для списка — положительное целое число, для `nil` — `0`, для точечных пар и точечных списков — `nil`.

Примеры:

- `(vl-list-length '())` возвращает `0`
- `(vl-list-length nil)` возвращает `0`

- `(vl-list-length '("a" "b" 1 56))` возвращает `4`
- `(vl-list-length '(10 . 12))` возвращает `nil`
- `(vl-list-length '(3 7 . 81))` возвращает `nil`

1.9.23. Функция *vl-member-if*

Проверяет истинность результата применения тест-функции хотя бы к одному элементу списка. В качестве тест-функций могут использоваться как обычные функции, так и `lambda`-функции.

```
(vl-member-if '<тест-функция> <список>)
```

Аргументы: `<тест-функция>` — одноместная функция, применяемая поэлементно ко второму аргументу; `<список>` — список с проверяемыми элементами.

Для передачи имени тест-функции можно использовать не только апостроф, но и функцию `function` *(см. разд. 1.16.7)*. При применении `function` апостроф не ставится.

Возвращаемое значение: остаток списка `<список>`, начиная с того элемента, для которого впервые тест-функция выдала результат, отличный от `nil`. Если для всех элементов аргумента `<список>` тест-функция вернула `nil`, то функция `vl-member-if` тоже возвращает `nil`.

Примеры:

- `(vl-member-if 'zerop '(4.6 0.0 0.0 9.2 0.0))` возвращает `(0.0 0.0 9.2 0.0)`
- `(vl-member-if 'numberp '(T "44.6"))` возвращает `nil`
- `(vl-member-if 'numberp '(T "44.6" 7))` возвращает `(7)`

1.9.24. Функция *vl-member-if-not*

Проверяет, возвращает ли тест-функция, примененная хотя бы к одному элементу списка, ложный результат.

```
(vl-member-if-not '<тест-функция> <список>)
```

Аргументы: `<тест-функция>` — одноместная функция, применяемая поэлементно ко второму аргументу; `<список>` — список с проверяемыми элементами.

Для передачи имени тест-функции можно использовать не только апостроф, но и функцию `function` *(см. разд. 1.16.7)*. При применении `function` апостроф не ставится.

Возвращаемое значение: остаток списка `<список>`, начиная с того элемента, для которого впервые тест-функция выдала результат, равный `nil`. Если для всех элементов аргумента `<список>` тест-функция вычисляет не `nil`, то функция `vl-member-if-not` возвращает `nil`.

Примеры:

- `(vl-member-if-not 'zerop '(0.0 0.0 12.0 0.0))` возвращает `(12.0 0.0)`
- `(vl-member-if-not 'numberp '(T "4.33"))` возвращает `(T "4.33")`
- `(vl-member-if-not 'numberp '(78 4.3))` возвращает `nil`

1.9.25. Функция *vl-position*

Вычисляет номер позиции элемента в списке.

```
(vl-position <элемент> <список>)
```

Аргументы: <элемент> — любой объект, проверяемый на присутствие в списке; <список> — список, в котором проверяется наличие объекта <элемент>.

Возвращаемое значение — целое число, являющееся номером позиции искомого элемента в списке (первый элемент списка имеет номер 0). Если <элемент> в аргументе <список> не обнаружен, то возвращается `nil`.

Примеры:

- `(vl-position 3 '(7 22 3 4 11 8))` возвращает `2`
- `(vl-position 5 '(7 22 3 4 11 8))` возвращает `nil`

1.9.26. Функция *vl-remove*

Удаляет элемент с заданным значением из списка.

```
(vl-remove <элемент> <список>)
```

Аргументы: <элемент> — любой объект, проверяемый на присутствие в списке; <список> — список, из которого нужно удалить объект <элемент>.

Возвращаемое значение — <список>, но без всех вхождений удаляемого объекта <элемент>. Если <элемент> в списке не обнаружен, то возвращается старое значение аргумента <список>.

Примеры:

- `(vl-remove 2 '(2 11 5 2 2 0))` возвращает `(11 5 0)`
- `(vl-remove 2 '(2 11 5 2 (2 0)))` возвращает `(11 5 (2 0))`
- `(vl-remove "2" '("2" "11" "5" "2" "2" "0"))` возвращает `("11" "5" "0")`
- `(vl-remove 5 '(1 2 16 2 8 1))` возвращает `(1 2 16 2 8 1)`

1.9.27. Функция *vl-remove-if*

Удаляет из списка все элементы, возвращающие `T` при проверке тест-функцией.

```
(vl-remove-if '<тест-функция> <список>)
```

Аргументы: `<тест-функция>` — функция, применяемая для проверки поочередно к каждому элементу списка; `<список>` — список, из которого нужно удалить объекты.

В качестве тест-функций могут использоваться также `lambda`-функции. Вместо апострофа возможно применение функции `function` *(см. разд. 1.16.7)*.

Возвращаемое значение — `<список>`, но без удаляемых элементов.

Примеры:

- `(vl-remove-if 'minusp '(-2 7 -11 -9 0 1))` возвращает `(7 0 1)`
- `(vl-remove-if (function minusp) '(-2 7 -11 -9 0 1))` возвращает `(7 0 1)`
- `(vl-remove-if 'listp '(10 nil 4 (7 8) 14))` возвращает `(10 4 14)`
- `(vl-remove-if 'listp '(10 nil 4 (7 . 8) 14))` возвращает `(10 4 14)`

1.9.28. Функция *vl-remove-if-not*

Удаляет из списка все элементы, возвращающие `nil` при проверке тест-функцией.

```
(vl-remove-if-not '<тест-функция> <список>)
```

Аргументы: `<тест-функция>` — функция, применяемая для проверки поочередно к каждому элементу списка; `<список>` — список, из которого нужно удалить объекты.

В качестве тест-функций могут использоваться также `lambda`-функции. Вместо апострофа возможно применение функции `function` *(см. разд. 1.16.7)*.

Возвращаемое значение — `<список>`, но без удаляемых элементов.

Примеры:

- `(vl-remove-if-not 'minusp '(-2 7 -11 -9 0 1))` возвращает `(-2 -11 -9)`
- `(vl-remove-if-not (function minusp) '(-2 7 -11 -9 0 1))` возвращает `(-2 -11 -9)`
- `(vl-remove-if-not 'listp '(10 nil 4 (7 8) 14))` возвращает `(nil (7 8))`
- `(vl-remove-if-not 'listp '(10 () 4 (7 8) 14))` возвращает `(nil (7 8))`
- `(vl-remove-if-not 'listp '(10 () 4 (7 . 8) 14))` возвращает `(nil (7 . 8))`

1.9.29. Функция *vl-some*

Проверяет поочередно элементы с одинаковым номером всех списков на предмет возвращения тест-функцией результата `T`.

```
(vl-some '<тест-функция> <список1> [<список2> ... [<списокN>]...])
```

Аргументы: <тест-функция> — функция, применяемая одновременно к элементам с одинаковыми номерами (к первым, ко вторым и т. д.) всех списков; <список1>, ... <списокN> — списки с проверяемыми элементами.

В качестве тест-функций могут использоваться также lambda-функции. Вместо апострофа возможно применение функции function *(см. разд. 1.16.7)*.

Возвращаемое значение — T, если тест-функция дала результат T хотя бы один раз (на элементах с одинаковыми номерами для всех списков). В противном случае возвращается nil.

Примеры:

- (vl-some '= '(7 8 0) '(0 3 0) '(9 5 0)) возвращает T
- (vl-some '/= '(7 8 0) '(0 3 0) '(9 5 0)) возвращает T
- (vl-some '> '(7 8 0) '(0 3 0) '(9 5 0)) возвращает nil
- (vl-some '>= '(7 8 0) '(0 3 0) '(9 5 0)) возвращает T

1.9.30. Функция *vl-sort*

Сортирует элементы в списке таким образом, чтобы тест-функция выдавала результат T после применения ее к новому списку. Для аргументов некоторых типов (например, целых) повторяющиеся элементы при перестановке могут удаляться.

```
(vl-sort <список> '<тест-функция>)
```

Аргументы: <список> — список, элементы которого сортируются; <тест-функция> — функция, которая может быть применена сразу ко всем элементам списка в качестве аргументов.

Возвращаемое значение — список, в котором все элементы взяты из аргумента <список>, но отсортированы с помощью функции <тест-функция>.

Примеры:

- (vl-sort '(0 4 2 1) '>) возвращает (4 2 1 0)
- (vl-sort '(0 4 0 2 2 1) '<) возвращает (0 1 2 4)
- (vl-sort '(0 0 0 0 0 0) '<) возвращает (0)
- (vl-sort '(0 0 0 0 0 0) '<=) возвращает (0)

При построении тест-функций могут фигурировать функции lambda и function *(см. разд. 1.16)*. Например,

```
(vl-sort '((3.2 -1.1) (5.9 6.0) (1.9 1.3)) (function (lambda (e1 e2)
(> (car e1) (car e2)))))
```

возвращает список ((5.9 6.0) (3.2 -1.1) (1.9 1.3)) с двумерными точками, упорядоченными по убыванию первой координаты.

1.9.31. Функция *vl-sort-i*

Вычисляет номера элементов в списке для сортировки элементов таким образом, чтобы тест-функция выдавала результат `T` после применения ее к новому списку. Повторяющиеся элементы при перестановке не удаляются (сравните с функцией `vl-sort`).

```
(vl-sort-i <список> '<тест-функция>)
```

Аргументы: `<список>` — список с номерами (целыми числами), указывающими порядок сортировки элементов; `<тест-функция>` — функция сортировки, которая может быть применена сразу ко всем элементам списка в качестве аргументов. Номер первого элемента списка равен 0.

Возвращаемое значение — список, в котором номера элементов указаны таким образом, чтобы выполнялась сортировка с помощью функции `<тест-функция>`.

Примеры:

- `(vl-sort-i '(0 4 2 1) '>)` возвращает `(1 2 3 0)`
- `(vl-sort-i '(0 4 0 2 2 1) '<)` возвращает `(2 0 5 4 3 1)`
- `(vl-sort-i '(0 0 0 0 0 0) '<)` возвращает `(5 4 3 2 1 0)`

Могут использоваться для алгоритма сортировки также пользовательские функции `lambda` и `function` *(см. разд. 1.16)*. Например,

```
(vl-sort-i '((3.2 -1.1) (5.9 6.0) (3.9 1.3)) (function (lambda (e1 e2)
(> (car e1) (car e2)))))
```

возвращает список `(1 2 0)`.

1.9.32. Функция *vl-catch-all-apply* и обработка ошибок

Передает для выполнения указанной функции список в качестве аргументов и дает возможность обработать прерывание, связанное с ошибкой (сравните с функцией `apply`, не предоставляющей такой возможности).

```
(vl-catch-all-apply '<функция> <список>)
```

Аргументы: `<функция>` — имя функции, которой передается список в качестве аргументов; `<список>` — список, элементы которого передаются как аргументы функции `<функция>`.

Возвращаемое значение — результат применения функции `<функция>` к списку `<список>`. Если применение функции `<функция>` приводит к ошибке, то возвращается специальный объект с характеристиками ошибки (см. описание функции `vl-catch-all-error-message`).

Примеры:

- `(vl-catch-all-apply '* '(2 -8 5))` возвращает `-80`
- `(vl-catch-all-apply 'sqrt '(-72))` возвращает `#<%catch-all-apply-error%>`, что является объектом типа VL-CATCH-ALL-APPLY-ERROR; далее для расшифровки объекта можно воспользоваться функцией `vl-catch-all-error-message`.

Функция `vl-catch-all-apply` может использоваться для квалифицированной обработки ошибок *(см. также специализированную функцию `*error*`)*. В качестве примера такой обработки *см. приложения 4 и 5*.

1.9.33. Функция *vl-catch-all-error-message*

Расшифровывает сообщение об ошибке, закодированное в объекте, которое при прерывании возвращает функция `vl-catch-all-apply`.

```
(vl-catch-all-error-message <объект>)
```

Аргумент — объект типа VL-CATCH-ALL-APPLY-ERROR, возвращенный функцией `vl-catch-all-apply`.

Возвращаемое значение — строка с текстом сообщения об ошибке.

Пример (связан с примером предыдущего раздела):

- `(setq err_ob (vl-catch-all-apply 'sqrt '(-72)))` возвращает `#<%catch-all-apply-error%>`
- `(vl-catch-all-error-message err_ob)` возвращает`"function undefined for argument: -72"` (в русской версии: `"функция для аргумента не задана: -72"`)

1.9.34. Функция *vl-catch-all-error-p*

Проверяет, является ли объект, возвращенный функцией `vl-catch-all-apply`, объектом типа VL-CATCH-ALL-APPLY-ERROR.

```
(vl-catch-all-error-p <аргумент>)
```

Тип аргумента — любой.

Возвращаемое значение: `T`, если `<аргумент>` относится к объектам типа VL-CATCH-ALL-APPLY-ERROR, и `nil`, если `<аргумент>` имеет другой тип.

Пример (связан с примером из *разд. 1.9.32*):

- `(setq err_ob (vl-catch-all-apply 'sqrt '(-72)))` возвращает `#<%catch-all-apply-error%>`
- `(vl-catch-all-error-p err_ob)` возвращает `T`

Для сравнения: `(type err_ob)` вернет `VL-CATCH-ALL-APPLY-ERROR`.

1.10. Функции ввода данных и указания объектов

В следующих разделах рассматриваются функции, которые дают возможность пользователю вводить данные в интерактивном режиме. Здесь же приводятся функции доступа к системной информации (переменным окружения, системным переменным и т. п.).

1.10.1. Функция *initget*

Задает ограничения последующего ввода данных.

```
(initget [<флаг>] [<строка>])
```

Вызов функции `initget` должен предшествовать в программе обращению к функциям `getint`, `getreal`, `getdist`, `getangle`, `getorient`, `getpoint`, `getcorner`, `getkword` (а также к рассматриваемым в *разд. 1.12* функциям `entsel`, `nentsel` и `nentselp`), поскольку именно для них устанавливаются ограничения.

В частности, `initget` может сохранять за пользователем возможность нажатия клавиши <Enter> вместо ввода данных (например, для выбора значения по умолчанию), а может и не сохранять. Отсутствие вызова функции `initget` перед обращением к функциям `getint`, `getreal` и т. п. означает, что в этом обращении нет ограничений на ввод данных.

Аргумент `<флаг>` функции `initget` — это битовый флаг, который должен быть целым числом (от 0 до 255), являющимся суммой битов с соответствующими каждому из них весовыми значениями (нумеруются по степеням числа 2). Биты описываются в табл. 1.4. Если бит не установлен, то его значение в битовом флаге считается равным нулю.

Таблица 1.4. *Значения битов функции* `initget`

Бит	Назначение
1	Не разрешает пустой ввод (*пустой ввод* — это нажатие только клавиши <Enter>)
2	Не разрешает ввод нуля
4	Не разрешает ввод отрицательных чисел
8	Разрешает указание точек за пределами лимитов, несмотря на значение системной переменной LIMCHECK
16	Данный бит не задействован
32	Заставляет AutoCAD использовать штриховую линию вместо сплошной для резиновой нити и резиновой рамки; если значение системной переменной POPUPS равно нулю, то AutoCAD игнорирует данный бит

Таблица 1.4 *(окончание)*

Бит	Назначение
64	Запрещает ввод третьей координаты для точки в функции `getdist`
128	Разрешает произвольный ввод в качестве ключевого слова; преобладает над другими битами и ключевыми словами; в частности, данный бит позволяет игнорировать значение бита 1

Примеры использования битового флага:

- `(initget 1)` — не разрешен пустой ввод
- `(initget 3)` — не разрешены пустой ввод и ввод нуля (3 = 1 + 2)
- `(initget 7)` — не разрешены пустой ввод, ввод нуля и ввод отрицательных чисел (7 = 1 + 2 + 4)
- `(initget 6)` — разрешен пустой ввод, но не разрешены ввод нуля и ввод отрицательных чисел (6 = 2 + 4)

Аргумент `<строка>` функции `initget` — это строка, ограниченная двойными кавычками, которая задает ключевые слова, являющиеся допустимыми вариантами ввода. В аргументе `<строка>` различные варианты ключевых слов разделяются одним или большим количеством пробелов, например: `"А Б В Г Д"`. Если пользователю при выборе ключевого слова разрешаются сокращенные варианты, тогда обязательная часть ключевого слова в аргументе `<строка>` должна быть указана в верхнем регистре, например: `"ПЕРесечение"`. В этом случае к допустимым относятся любые следующие варианты ввода: `"ПЕР"`, `"пер"`, `"ПЕРЕ"`, `"пере"`, `"перес"`, `"ПЕРЕСЕЧЕН"`, `"ПЕРЕСЕЧЕНИЕ"` (после трех обязательных букв может идти любое количество необязательных букв, вплоть до набора полного ключевого слова; при вводе букв верхний и нижний регистры равноправны); недопустимыми вариантами будут `"П"`, `"ПЕ"`, `"п"`, `"пе"`.

Возможны случаи, когда вариант сокращенного ввода начинается не с первой буквы ключевого слова, например:

```
(initget "выХод")
```

Здесь допустимыми вариантами ввода будут `"х"`, `"хо"`, `"ход"` или `"выход"`, `"вых"`, `"выхо"`.

Если ключевое слово в аргументе `<строка>` указано только в верхнем или только в нижнем регистре, то рядом через запятую можно указать сокращенный вариант: `"КРАСНЫЙ,КР"` — это равносильно записи `"КРасный"`.

В локализованных (т. е. переведенных на другие языки) версиях системы AutoCAD в аргументе `<строка>` могут присутствовать варианты ввода как для локализованной, так и для английской версий. Тогда в начале должны идти локализованные ключевые слова, а затем их английские аналоги, причем первый анг-

лийский аналог обязан начинаться с символа подчеркивания. Количество локализованных ключевых слов должно равняться количеству английских ключевых слов. Например: "Прямой Криволинейный _Straight Curved". В этом случае даже при вводе русского варианта будет возвращаться английское значение ("Straight" или "Curved").

Сама функция initget всегда возвращает nil. Однако за ней (не обязательно в следующей строке) должна идти функция, для которой функция initget установила ограничения ввода. В качестве возвращаемого значения эта функция (например, getreal) выдаст допустимое значение, указанное пользователем, причем в случае ввода сокращенного варианта ключевого слова — полный вариант ключевого слова.

Поскольку не все биты функции initget используются со всеми функциями ввода данных, то в табл. 1.5 разбираются варианты применения (в случае возможности использования ставится плюс).

Таблица 1.5. *Биты функции* `initget` *и их использование в функциях ввода*

Функция	Бит 1	Бит 2	Бит 4	Бит 8	Бит 32	Бит 64	Бит 128
getint	+	+	+				+
getreal	+	+	+				+
getdist	+	+	+		+	+	+
getangle	+	+			+		+
getorient	+	+			+		+
getpoint	+			+	+		+
getcorner	+			+	+		+
getkword	+						+

Функции entsel, nentsel и nentselp не используют биты функции initget и управляются только ключевыми словами.

Функция getstring не использует ни биты, ни ключевые слова функции initget.

1.10.2. Функция *getkword*

Дает возможность пользователю ввести допустимое ключевое слово.

```
(getkword [<запрос>])
```

Аргумент <запрос> — произвольная текстовая строка.

Возвращаемое значение (ключевое слово или nil) зависит от допустимых ключевых слов и битов, установленных соответствующим предварительным вызовом функции initget.

В листинге 1.9 рассматривается пример совместной работы функций `initget` и `getkword`.

Листинг 1.9. Пример использования функции `getkword`

```
; Глава 01\Листинг 1.09.lsp
(initget 1 "Да Нет")
(setq reply (getkword "Преобразовать линию? [Да/Нет]: "))
```

В этом примере функция `getkword` выводит запрос `"Преобразовать линию? [Да/Нет]: "` и ожидает ответа пользователя. По окончании ввода пользователем допустимого ответа LISP присваивает переменной `reply` значение `"Да"` или `"Нет"`. Если введен недопустимый ответ, то генерируется сообщение об ошибке, и снова повторяется предыдущий запрос.

Пустой ввод не допускается (хотя, изменив значение битового флага, можно было бы разрешить пустой ввод и далее использовать этот вариант для выбора значения по умолчанию).

1.10.3. Функция *getint*

Дает возможность пользователю ввести целое число (в диапазоне от –32 768 до +32 767) или допустимое ключевое слово.

```
(getint [<запрос>])
```

Аргумент `<запрос>` — произвольная текстовая строка.

Возвращаемое значение — целое число, ключевое слово или `nil`. Зависит от допустимых ключевых слов и битов, установленных соответствующим предварительным вызовом функции `initget`.

В листингах 1.10 и 1.11 рассматривается пример совместной работы функций `initget` и `getint`.

Листинг 1.10. Пример 1 использования функции `getint`

```
; Глава 01\Листинг 1.10.lsp
(initget 6)
(setq num (getint "Введите номер детали <1>: "))
```

В этом примере сначала устанавливаются ограничения на будущий ввод целого числа (не допускаются нуль и отрицательные числа). Разрешен пустой ввод. Функция `getint` выводит на экран запрос `"Введите номер детали <1>: "`. Ее возвращаемым значением будет `nil`, если пользователь ответит простым нажатием клавиши <Enter>, или введенное пользователем допустимое целое число (например, `21`). В случае ввода пользователем недопустимого целого числа (например, `0`

или -5) функция getint выдаст сообщение об ошибочном значении и будет ожидать допустимого варианта ввода.

В программе, приведенной в листинге 1.10, далее должна быть предусмотрена обработка пустого ввода для того, чтобы в этом случае присвоить переменной num значение по умолчанию (1):

```
(if (not num) (setq num 1))
```

Листинг 1.11. Пример 2 использования функции getint

```
; Глава 01\Листинг 1.11.lsp
(initget 6 "Левый Правый Нижний Верхний")
(setq num (getint "Номер или [Левый/Правый/Нижний/Верхний] <2>: "))
```

В этом примере устанавливаются ограничения на будущий ввод целого числа (не допускаются нуль и отрицательные числа), однако разрешены пустой ввод и ввод одного из четырех ключевых слов. Далее в программе должен идти анализ значения переменной num на nil, на положительное целое число и на ключевые слова ("Левый", "Правый", "Нижний", "Верхний").

1.10.4. Функция *getreal*

Дает возможность пользователю ввести вещественное число или допустимое ключевое слово.

```
(getreal [<запрос>])
```

Аргумент <запрос> — произвольная текстовая строка.

Возвращаемое значение — вещественное число, ключевое слово или nil. Зависит от допустимых ключевых слов и битов, установленных соответствующим предварительным вызовом функции initget.

Пример:

♦ `(setq x2 (getreal "Вторая координата: "))`

1.10.5. Функция *getdist*

Получает в интерактивном режиме расстояние или просто вещественное число с помощью ввода этого числа (на клавиатуре в текущих линейных единицах) или с помощью двух точек (указанных мышью или в координатах на клавиатуре). Выбор варианта ввода (числом или точками) предоставляется пользователю.

```
(getdist [<точка1>] [<запрос>])
```

Типы аргументов: <точка1> — список из двух или трех чисел (координаты двумерной или трехмерной точки), <запрос> — произвольная текстовая строка.

Вариант ввода числа в ответ на запрос функции getdist доступен пользователю, независимо от наличия аргумента <точка1>.

Если аргумент <точка1> задан, то расстояние вычисляется либо с помощью ввода числа (на клавиатуре), либо указанием одной точки, до которой будет рассчитано расстояние от точки, заданной аргументом <точка1>.

Если аргумент <точка1> не задан, то расстояние вычисляется либо с помощью ввода числа (на клавиатуре), либо с помощью указания двух точек.

Возвращаемое значение — вещественное число (в текущей форме линейных единиц), ключевое слово или nil. Зависит от допустимых ключевых слов и битов, установленных соответствующим предварительным вызовом функции initget.

Пример:

♦ `(setq dse (getdist '(15.0 27.2) "Новое расстояние: "))`

1.10.6. Функция *getangle*

Получает в интерактивном режиме угол (как вещественное число в радианах) с помощью ввода числа (на клавиатуре в текущих угловых единицах) или с помощью двух точек (уазанных мышью или в координатах на клавиатуре). Выбор варианта ввода (числом или точками) предоставляется пользователю.

```
(getangle [<точка1>] [<запрос>])
```

Типы аргументов: <точка1> — список из двух или трех чисел (координаты двумерной или трехмерной точки), <запрос> — произвольная текстовая строка.

Возвращаемое значение — вещественное число, ключевое слово или nil. Зависит от допустимых ключевых слов и битов, установленных соответствующим предварительным вызовом функции initget.

Вариант ввода числа в ответ на запрос функции getangle доступен пользователю, независимо от наличия аргумента <точка1>.

Если аргумент <точка1> задан, то угол вычисляется либо с помощью ввода числа (на клавиатуре в радианах), либо с помощью указания одной точки (мышью или на клавиатуре). В последнем случае рассчитывается угол, который относительно нулевого направления (обычно это положительное направление оси X) образует отрезок, идущий из точки <точка1> до точки, указанной пользователем.

Если аргумент <точка1> не задан, то угол вычисляется либо с помощью ввода числа (на клавиатуре в радианах), либо с помощью указания двух точек. В последнем случае рассчитывается угол, который образует отрезок, идущий из первой указанной пользователем точки во вторую.

При вводе угла с помощью числа он принимается с учетом значения системных переменных ANGBASE и ANGDIR; возвращаемое значение учитывает значение ANGBASE и игнорирует значение ANGDIR.

Пример:

♦ `(setq ange (getangle '(15.0 27.2) "Угол между ними: "))`

1.10.7. Функция *getorient*

Получает угол как функция `getangle`, однако возвращаемое значение не учитывает значений системных переменных ANGBASE и ANGDIR (т. е. как будто ANGBASE=0, ANGDIR=0).

```
(getorient [<точка1>] [<запрос>])
```

Типы аргументов: `<точка1>` — список из двух или трех чисел (координаты двумерной или трехмерной точки), `<запрос>` — произвольная текстовая строка.

Возвращаемое значение — вещественное число, ключевое слово или `nil`. Зависит от допустимых ключевых слов и битов, установленных соответствующим предварительным вызовом функции `initget`.

Пример:

♦ `(setq ango (getorient '(15.0 27.2) "Следующий угол: "))`

1.10.8. Функция *getpoint*

Дает возможность пользователю ввести точку с помощью клавиатуры или мыши. Если задан аргумент `<точка1>`, то при вводе точки с помощью мыши рисуется резиновая нить, начинающаяся в точке, заданной аргументом `<точка1>`.

```
(getpoint [<точка1>] [<запрос>])
```

Типы аргументов: `<точка1>` — список из двух или трех чисел (координаты двумерной или трехмерной точки), `<запрос>` — текстовая строка.

Возвращаемое значение — список (координаты трехмерной точки в текущей системе координат), ключевое слово или `nil`. Зависит от допустимых ключевых слов и битов, установленных соответствующим предварительным вызовом функции `initget`.

Пример:

♦ `(setq pt0 (getpoint '(15.0 27.2) "Следующая точка: "))`

1.10.9. Функция *getcorner*

Дает возможность пользователю ввести точку с помощью клавиатуры или мыши. При вводе точки с помощью мыши рисуется прямоугольная резиновая рамка, начинающаяся в точке, заданной аргументом `<точка1>` (этот аргумент не может быть опущен).

```
(getcorner <точка1> [<запрос>])
```

Типы аргументов: <точка1> — список из двух или трех чисел (координаты двумерной или трехмерной точки), <запрос> — произвольная текстовая строка.

Возвращаемое значение — список (координаты трехмерной точки в текущей системе координат), ключевое слово или nil. Зависит от допустимых ключевых слов и битов, установленных соответствующим предварительным вызовом функции initget.

Пример:

- (setq pt1 (getpoint '(15.0 27.2) "Задай точку: "))

1.10.10. Функция *getstring*

Дает возможность пользователю ввести строку с клавиатуры. Если задан аргумент <флаг> и он отличен от nil, то в строке, вводимой пользователем, допускаются пробелы (признаком конца является нажатие клавиши <Enter>). Если аргумент <флаг> отсутствует или задан равным nil, то признаком конца строки является пробел или нажатие клавиши <Enter>.

(getstring [<флаг>] [<запрос>])

Типы аргументов: <флаг> — любое значение (рекомендуется использовать только T или nil), <запрос> — текстовая строка.

Возвращаемое значение — строка (в случае пустого ввода возвращается строка нулевой длины, т. е. ""). Работа функции getstring не связана с предварительным вызовом функции initget.

Пример:

- (setq str1 (getstring T "Назови объект (пробелы допускаются): "))

1.10.11. Функция *getvar*

Читает текущее значение системной переменной с наименованием, заданным аргументом <имя>.

(getvar <имя>)

Тип аргумента: текстовая строка (в верхнем или нижнем регистрах) с наименованием системной переменной AutoCAD.

Возвращаемое значение — текущее значение указанной системной переменной в текущем рисунке (если задано недопустимое имя переменной, то возвращается nil).

Примеры:

- (getvar "CECOLOR") возвращает строку — значение системной переменной CECOLOR (имя текущего цвета)

- `(getvar "OSMODE")` возвращает целое число — значение системной переменной OSMODE (текущий режим объектной привязки)
- `(getvar "NO")` возвращает `nil`

1.10.12. Функция *setvar*

Присваивает новое текущее значение системной переменной с наименованием, заданным аргументом `<имя>`. Данная функция не может изменять значения системных переменных, которые имеют пометку "только чтение" ("read only").

```
(setvar <имя> <значение>)
```

Типы аргументов: `<имя>` — текстовая строка (в верхнем или нижнем регистрах) с наименованием системной переменной AutoCAD, `<значение>` — число, список или строка (конкретный тип определяется именем переменной).

Возвращаемое значение — новое значение указанной системной переменной. В случае попытки присвоить недопустимое значение или использовать имя несуществующей переменной выдается сообщение об ошибке.

Примеры:

- `(setvar "CECOLOR" "5")` возвращает `"5"` — новое значение системной переменной CECOLOR (номер текущего цвета)
- `(setvar "CECOLOR" "RGB:251,155,142")` возвращает `"RGB:251,155,142"`
- `(setvar "CECOLOR" 5)` выдает сообщение: ***; error: AutoCAD variable setting rejected: "CECOLOR" 5*** *(; ошибка: Настройка системной переменной AutoCAD отвергнута: "CECOLOR" 5)*

1.10.13. Функция *getenv*

Читает текущее значение переменной окружения AutoCAD или Windows с наименованием, заданным аргументом `<имя>`.

```
(getenv <имя>)
```

Тип аргумента: текстовая строка (в верхнем или нижнем регистрах) с наименованием переменной окружения. Под переменными окружения понимаются значения параметров настроек AutoCAD, хранящиеся в реестре в профиле текущего пользователя системы, и переменные системного окружения Windows.

При вызове функции `getenv` система AutoCAD сначала пытается найти параметр с наименованием `<имя>` в разделе реестра `HKEY_CURRENT_USER\Software\Autodesk\AutoCAD\<версия>\<номер>\FixedProfile` (просматриваются подразделы `Editor Configuration`, `General` и `General Configuration`). В качестве составляющей `<версия>` должны фигурировать обозначения версии (`R16.1`, `R16.2` и т. д.), а в качестве `<номер>` — номер сборки (`ACAD-4001:409` и т. п.).

При этом для подразделов Editor Configuration (параметры[3] AutoSnapColor, AutoSnapControl, AutoSnapPolarAddAng, AutoSnapPolarAng, AutoSnapPolarDistance, ObjectSnapCursorSize, PickAdd, PickDrag, SDF_AttributeExtractTemplateFile) и General Configuration (параметры CommandDialogs, DYNMODE, DYNPICOORDS, DYNPIFORMAT) обязательно соблюдение регистра. Для некоторых параметров подраздела General (acedChangeCmdEchoWithoutUndo, ANSIHatch, ANSILinetype, ISOHatch, ISOLinetype, LastTemplate, ProfileStorage, StartUpType) соблюдение регистра не обязательно. Однако параметры InsertUnitsDefSource, InsertUnitsDefTarget, Measureinit этого подраздела тоже являются регистрозависимыми.

Параметр TaskBar подраздела General Configuration не доступен через функцию getenv (для изменения следует пользоваться командой TASKBAR).

Если переменная <имя> не найдена в реестре, то поиск продолжается в списке переменных окружения Windows (регистр в наименовании такой переменной не важен).

Возвращаемое значение — значение указанной переменной (если в окружении такая переменная не обнаружена или ее значение не задано, то возвращается nil).

Пример:

- (getenv "PickDrag") возвращает значение переменной окружения AutoCAD (например, "1")
- (getenv "pickdrag") возвращает nil (не соблюден регистр в названии переменной)
- (getenv "StartupType") возвращает "Scratch"
- (getenv "STARTUPTYPE") возвращает "Scratch" (регистр не важен)
- (getenv "TEMP") возвращает значение переменной окружения TEMP (например, "C:\\TEMP")

1.10.14. Функция *setenv*

Присваивает новое значение переменной окружения AutoCAD или Windows с наименованием, заданным аргументом <имя>.

```
(setenv <имя> <значение>)
```

Типы аргументов: текстовые строки. Требования по соблюдении регистра в некоторых переменных окружения приведены в предыдущем разделе.

Возвращаемое значение — значение аргумента <значение>.

Пример:

- (setenv "MY_ENVVAR" "SPACE") возвращает "SPACE" (в окружении создается новая переменная)

[3] Списки параметров приведены по версии AutoCAD 2006.

Для того чтобы сбросить значение переменной окружения, ей следует присвоить значение "" (после этого функция getenv в качестве ее значения выдаст nil).

1.10.15. Функция *getcfg*

Извлекает данные приложения из раздела AppData файла конфигурации (например, для версии 2006 это файл acad2006.cfg) текущего пользователя.

```
(getcfg <строка>)
```

Строка, заданная аргументом <строка>, должна иметь вид "AppData/имя_приложения /имя_раздела/ … /имя_параметра".

Тип возвращаемого значения — строка или nil.

Пример:

- (getcfg "AppData/My/Two/Quattro")

1.10.16. Функция *setcfg*

Записывает данные приложения в раздел AppData файла конфигурации (например, acad2006.cfg) текущего пользователя.

```
(setcfg <имя> <значение>)
```

Аргумент <имя> должен быть строкой вида "AppData/имя_приложения/имя_раздела / … /имя_параметра" (допустимая длина — до 496 символов). Аргумент <значение> — строка длиной до 512 символов (более длинные строки затем не могут быть прочитаны функцией getcfg).

Возвращаемое значение — значение аргумента <значение> или nil (в случае ошибки).

Пример:

- (setcfg "AppData/My/Two/Quattro" "123") возвращает "123"

1.10.17. Функция *getcname*

Получает имя команды в локализованной или английской версии AutoCAD.

```
(getcname <имя>)
```

Тип аргумента: строковый.

Возвращаемое значение — строка или nil.

Примеры (для русской версии):

- (getcname "_LINE") возвращает "ОТРЕЗОК"
- (getcname "ОТРЕЗОК") возвращает "_LINE"

1.10.18. Функция *getfiled*

Вызывает диалоговое окно выбора файла.

```
(getfiled <заголовок> <имя> <расширение> <флаг>)
```

Аргумент `<заголовок>` используется как заголовок диалогового окна; аргумент `<имя>` — это имя искомого файла или папки, с которой предлагается начать поиск; аргумент `<расширение>` — расширение отображаемых файлов (`<расширение>`, равное `""`, заменяется на `"*"`); аргумент `<флаг>` — опции функции *(см. ниже)*.

Типы аргументов: строковые (кроме последнего, который должен быть целым числом).

Возвращаемое значение — строковое (имя файла, выбранного пользователем) или `nil`. Имя файла может быть с путем или без него, в зависимости от других аргументов *(см. далее описание аргумента `<флаг>`)*.

Второй аргумент функции `getfiled` может задавать либо имя файла по умолчанию, либо папку, с которой предлагается начать поиск. Если значение аргумента `<имя>` заканчивается на обратную косую черту (`"\\"`) или косую черту (`"/"`), то оно рассматривается как имя папки. Пример:

```
(getfiled "Выбор файла C++" "g:\\Programs\\ObjectARX\\" "cpp" 0)
```

Это выражение открывает окно, приведенное на рис. 1.6.

Если в приведенном примере опустить последнюю черту, то система будет рассматривать значение аргумента как имя файла. Пример:

```
(getfiled "Выбор файла C++" "g:\\Programs\\ObjectARX\\Book15" "cpp" 0)
```

Но если такой файл не существует, то в окне откроется папка по умолчанию (обычно это папка Мои документы текущего пользователя). Длина значения аргумента `<имя>` зависит от платформы и колеблется от 78 до 511 символов.

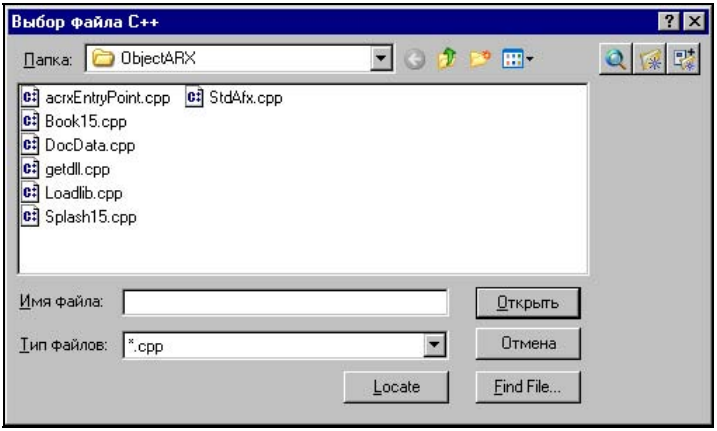


Рис. 1.6. Диалоговое окно выбора файла

Третий аргумент определяет то расширение, которое выводится в поле **Тип файлов** (Files of type). Окно выбора отображает файлы только выбранного расширения.

Пользователь может задать сразу несколько расширений. В этом случае они должны отделяться друг от друга точкой с запятой, например: `"dwg;bmp;jpg"`. При открытии окна система показывает в поле **Тип файлов** (Files of type) первое из расширений.

Если в качестве значения аргумента `<расширение>` задается `"dwg"`, то в окне справа появляется область просмотра (рис. 1.7).

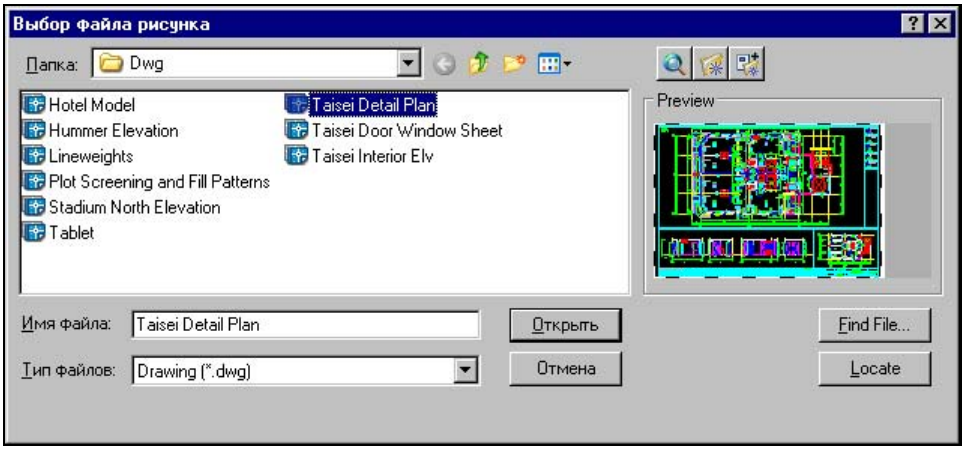


Рис. 1.7. Диалоговое окно выбора файла с областью просмотра

Когда требуется отразить в окне все файлы папки, тогда необходимо задавать в качестве расширения `"*"` или `""`.

Битовый флаг, задаваемый целочисленным аргументом `<флаг>`, получается как сумма тех битов, нумеруемых степенями числа 2, которые будут установлены. Описание битов приведено в табл. 1.6.

Таблица 1.6. *Значения битов функции* `getfiled`

Бит	Значение
1	Если бит установлен, то название кнопки **Открыть** (см. рис. 1.6) заменяется на **Сохранить**. При нажатии кнопки выводится предупреждение о том, что такой файл уже существует и будет заменен. Если бит не установлен, то кнопка закрытия окна называется **Открыть**. При нажатии кнопки выводится предупреждение, если файл с введенным именем не существует (необходимо ввести правильное имя)
2	Не используется

Таблица 1.6 *(окончание)*

Бит	Значение
4	Если бит установлен, то разрешается вводить имя файла с любым расширением (или без расширения). Если бит не установлен, то допускается имена либо без расширения, либо только с расширениями, указанными в аргументе `<расширение>`
8	Бит имеет значение, когда в аргументе `<имя>` не задан путь — в этом случае система AutoCAD выполняет поиск файла `<имя>` по списку путей поддержки, заданных в настройке. Возвращается имя найденного файла: с путем, если бит 1 не установлен, и без пути, если бит установлен
16	Если бит установлен, то аргумент `<имя>` трактуется как имя папки, в которой нужно искать файл; при этом в поле **Имя файла** (File name) ничего не вписывается по умолчанию (см. рис. 1.6). Аналогичный эффект достигается и в том случае, если аргумент `<имя>` заканчивается чертой
32	Если бит установлен, то в случае одновременной установки бита 1 не будет выводиться предупреждение о записи поверх существующего файла
64	Установка бита не разрешает использование удаленных файлов (с URL-адресами в имени)
128	Если бит установлен, то URL-адреса запрещены

1.10.19. Функция *acad_colordlg*

Вызывает диалоговое окно с палитрой, в котором пользователь должен выбрать цвет в диапазоне от 0 до 256.

```
(acad_colordlg <номер> [<флаг>])
```

Аргумент `<номер>` показывает в окне цвет, предлагаемый в качестве значения по умолчанию. Если аргумент `<флаг>` не опущен и задан равным `nil`, то в окне гасятся кнопки **ByLayer** (ПоСлою) и **ByBlock** (ПоБлоку).

Типы аргументов: `<номер>` — целое число в диапазоне 0—256 (0 соответствует значению ByBlock (ПоБлоку), 256 — ByLayer (ПоСлою)), `<флаг>` — любое значение (дополнительно влияет только `nil`).

Возвращаемое значение — целое число (номер цвета, выбранного пользователем). Если пользователь нажмет кнопку **Cancel** (Отмена), то возвращается `nil`.

Пример:

- `(acad_colordlg 5)` вызывает диалоговое окно **Select Color** (Выбор цвета), приведенное на рис. 1.8.

В качестве значения цвета по умолчанию на рис. 1.8 предлагается цвет `"blue"`, имеющий номер 5. Окно открывается только с одной вкладкой. Еще две вкладки доступны при использовании более новой функции `acad_truecolordlg`.

Рис. 1.8. Диалоговое окно **Select Color** с одной вкладкой

1.10.20. Функция *acad_truecolordlg*

Вызывает диалоговое окно **Select Color** (Выбор цвета) с полной палитрой цветов, в котором пользователь должен выбрать цвет.

```
(acad_truecolordlg <цвет> [<флаг>])
```

Аргумент `<цвет>` показывает в окне цвет, предлагаемый в качестве значения по умолчанию. Если аргумент `<флаг>` не опущен и задан равным `nil`, то в окне на вкладке **Index Color** (Номер цвета) не доступны кнопки **ByLayer** (ПоСлою) и **ByBlock** (ПоБлоку).

Аргумент `<цвет>` может быть либо целым числом в диапазоне 0—256 (0 соответствует значению ByBlock (ПоБлоку), 256 — ByLayer (ПоСлою)), либо точечной парой с DXF-кодом 62, 420 или 430. Примеры допустимых точечных пар в аргументе `<цвет>`: `(62 . 6)`, `(420 . 2844372)`, `(430 . "PANTONE(R) pastel coated$PANTONE Blue 0821 C")`.

С DXF-кодом 62 в качестве значения используются целые числа от 0 до 256.

С DXF-кодом 420 используются длинные целые номера, отражающие цвет, составленный из RGB-компонент. Например, цвету "RGB:43,102,212" соответствует число 2844372 = 43 * 16^4 + 102 * 16^2 + 212.

С DXF-кодом 430 используются названия из текущего альбома цветов.

Возвращаемое значение — список с точечными парами, соответствующими цвету, выбранному пользователем в диалоговом окне. Если пользователь нажмет кнопку **Cancel** (Отмена), то возвращается `nil`.

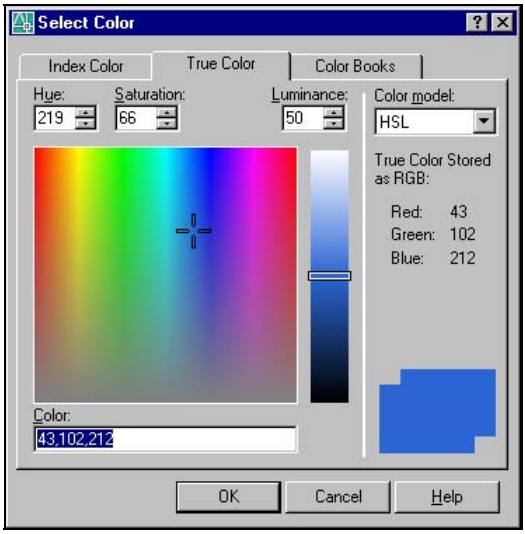


Рис. 1.9. Диалоговое окно **Select Color** с активной вкладкой **True Color**

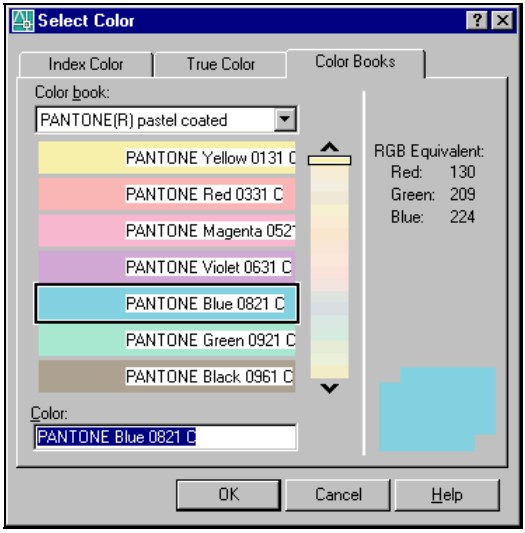


Рис. 1.10. Диалоговое окно **Select Color** с активной вкладкой **Color Books**

Примеры (в предположении, что пользователь во всех случаях нажал клавишу <Enter>, соглашаясь со значением по умолчанию):

- `(acad_truecolordlg 5)` возвращает `((62 . 5))`
- `(acad_truecolordlg '(62 . 5))` возвращает `(62 . 5)`
- `(acad_truecolordlg '(420 . 2844372))` возвращает `((62 . 152) (420 . 2844372))`

- `(acad_truecolordlg '(430 . "PANTONE(R) pastel coated$PANTONE Blue 0821 C"))` возвращает `((62 . 141) (420 . 8573408) (430 . "PANTONE(R) pastel coated$PANTONE Blue 0821 C"))`

На рис. 1.9 и 1.10 приведен вид окна **Select Color** (Выбор цвета), открываемого в двух последних примерах.

1.10.21. Функция *acad_truecolorcli*

Запрашивает цвет из полной палитры, в режиме командной строки.

```
(acad_truecolorcli <цвет> [<флаг>] [<подсказка>])
```

Аргумент `<цвет>` показывает цвет, предлагаемый в качестве значения по умолчанию. Если аргумент `<флаг>` не опущен и задан равным `nil`, то при ответе пользователя не доступны значения ByLayer (ПоСлою) и ByBlock (ПоБлоку).

Необязательный аргумент `<подсказка>` — строка с дополнительной подсказкой (в качестве подсказки по умолчанию выводится **New color** (Новый цвет)). Однако использование этого аргумента в версиях 2005—2006 не реализовано.

Допустимые значения первого аргумента аналогичны значениям первого аргумента функции `acad_truecolordlg`.

Пример:

- `(acad_truecolorcli '(420 . 2844372) "Наружный цвет")`

1.11. Функции печати, вывода сообщений и доступа к файлам

В данном разделе рассматриваются функции, выполняющие операции с файлами (поиск, открытие и закрытие), операции чтения и записи строк или символов в файл, а также операции вывода сообщений на экран. Здесь же рассматриваются близкие по духу функции `vl-prin1-to-string` и `vl-princ-to-string`.

1.11.1. Функция *findfile*

Ищет файл по имени (краткому или полному).

```
(findfile <файл>)
```

Если аргумент содержит имя файла без полного пути, то поиск выполняется сначала в текущем каталоге (обычно это папка Мои документы текущего пользователя), а затем по стандартным для системы AutoCAD путям поиска файлов. Возвращается полное имя файла, если файл найден, или `nil` — в противном случае. Аргумент должен быть текстовой строкой.

Примеры (для версии 2006):

- `(findfile "Support")` возвращает `"d:\\AutoCAD 2006\\Support"`
- `(findfile "acad.cui")` возвращает `"C:\\Documents and Settings\\ … \\Application Data\\Autodesk\\AutoCAD 2006\\R16.2\\enu\\support\\acad.cui"`

1.11.2. Функция *open*

Открывает файл.

```
(open <файл> <режим>)
```

Аргументы: `<файл>` — имя открываемого файла; `<режим>` — это строка из одного символа, определяющего режим, в котором открывается файл (`"w"` или `"W"` — запись, `"a"` или `"A"` — дополнение, `"r"` или `"R"` — чтение).

Если аргумент `<файл>` указывает на несуществующий файл в режимах `"w"`, `"W"`, `"a"` и `"A"`, то он создается. Если указано неполное имя (без пути), то файл ищется в текущем каталоге и по списку путей файлов поддержки.

Возвращаемое значение — дескриптор файла (при успешном открытии) или `nil` (при неудаче).

Пример:

- `(setq fo (open "Dim.txt" "w"))` возвращает `#<file "Dim.txt">` (дескриптор файла)

Дескриптор будет использоваться в операциях чтения-записи файла, поэтому его (дескриптор) обязательно нужно сохранять в какой-нибудь переменной.

1.11.3. Функция *close*

Закрывает файл.

```
(close <дескриптор>)
```

Единственным аргументом является `<дескриптор>` — переменная, создаваемая функцией `open` и управляющая процессами записи и чтения.

Функция `close` возвращает `nil`, если указан действительный дескриптор, или сообщение об ошибке.

Пример (продолжение примера из *разд. 1.11.2*):

- `(close fo)` возвращает `nil`

1.11.4. Функция *read-line*

Читает строку из файла с заданным дескриптором.

```
(read-line [<дескриптор>])
```

Если аргумент `<дескриптор>` опущен, то выполняется чтение строки с клавиатуры (строка должна закончиться нажатием клавиши <Enter>).

Возвращаемое значение — текстовая строка, прочитанная из файла или с клавиатуры.

Дескриптор содержит внутри себя информацию о текущей записи (т. е. той строке, с которой будет выполняться очередная операция чтения или записи). Сразу же после выполнения функции `open` дескриптор указывает на первую запись файла.

Пример применения функции `read-line` *см. в разд. 1.11.28*.

1.11.5. Функция *write-line*

Записывает строку, заданную аргументом `<строка>`, в файл с дескриптором, заданным аргументом `<дескриптор>`. Если аргумент `<дескриптор>` опущен или равен `nil`, то система AutoCAD выводит строку на экран. При записи строка, заданная аргументом `<строка>`, заносится в файл (или выводится на экран) без ограничивающих ее двойных кавычек.

```
(write-line <строка> [<дескриптор>])
```

Возвращаемое значение — значение аргумента `<строка>`.

Пример:

- `(write-line "Плазовая книга" fo)` возвращает `"Плазовая книга"`

Замечание

Если выполнить одно только выражение, приведенное в примере, с выводом на экран, то в командной строке к тексту без кавычек добавится еще возвращаемое значение — тот же текст в кавычках.

1.11.6. Функция *princ*

Записывает значение аргумента `<аргумент>`, который может быть любым выражением LISP (а также числом, списком, строкой, именем примитива и т. д.), в виде текстовой строки в файл с заданным дескриптором.

```
(princ [<аргумент> [<дескриптор>]])
```

Если аргумент `<дескриптор>` опущен или равен `nil`, то `<аргумент>` выводится на экран. Если `<аргумент>` — строка, то при записи она заносится в файл (или выводится на экран) без ограничивающих ее двойных кавычек.

Возвращаемое значение — значение аргумента `<аргумент>` (для вещественных чисел выводятся по шесть значащих цифр). Если опущены оба аргумента, то функция осуществляет так называемый *тихий выход*, т. е. не возвращает никакого значения, в том числе `nil` (чтобы не загромождать командную строку ненужными данными и сообщениями).

Примеры:

- `(princ '(4.5678123 11.1) fo)` пишет в файл и возвращает `(4.56781 11.1)`
- `(princ "Мороз" fo)` пишет в файл `Мороз` и возвращает `"Мороз"`
- `(princ)` не возвращает ничего

1.11.7. Функция *prin1*

Функция аналогична `princ`. Но если `<аргумент>` — строка, то при записи она заносится в файл (или выводится на экран) с ограничивающими двойными кавычками; специальные символы (обратная косая черта, кавычки, конец строки и т. п.) предваряются обратной косой чертой.

```
(prin1 [<аргумент> [<дескриптор>]])
```

Возвращаемое значение — значение аргумента `<аргумент>`. Если опущены оба аргумента, то функция `prin1` тоже осуществляет тихий выход.

Примеры:

- `(prin1 '(4.5678123 11.1) fo)` пишет в файл и возвращает `(4.56781 11.1)`
- `(prin1 "Мороз и солнце" fo)` пишет в файл и возвращает `"Мороз и солнце"`
- `(prin1)` не возвращает ничего

1.11.8. Функция *print*

Функция аналогична `prin1`, но при выводе добавляет перед значением аргумента `<аргумент>` символ конца строки предыдущей записи, а после значения — пробел.

```
(print [<аргумент> [<дескриптор>]])
```

Возвращаемое значение — вычисленное значение аргумента `<аргумент>`. Функция тоже осуществляет тихий выход, если оба аргумента опущены.

Примеры:

- `(print '(4.5678123 11.1) fo)` пишет в файл `\n(4.56781 11.1)_` и возвращает `(4.56781 11.1)` (перевод строки обозначен символами `\n`, а пробел в конце — символом подчеркивания)
- `(print "Мороз" fo)` пишет в файл `\n"Мороз"_` и возвращает `"Мороз"`
- `(print)` не возвращает ничего

1.11.9. Функция *read-char*

Читает символ из файла. Если аргумент не задан, то выполняет чтение символа из буфера клавиатуры.

```
(read-char [<дескриптор>])
```

Возвращает целое число — код прочитанного символа (аналогично функции `ascii`). При чтении из буфера клавиатуры — для клавиши <Enter> функция возвращает 10.

См. пример в *разд. 1.11.28*.

1.11.10. Функция *write-char*

Записывает буквенно-цифровой символ с заданным кодом в файл.

```
(write-char <код> [<дескриптор>])
```

Если аргумент `<дескриптор>` не задан, то выводит символ на экран.

Возвращаемое значение — целое число (код выведенного символа). Невозможно с помощью данной функции записать символ с кодом 0.

Примеры:

- `(write-char 50)` возвращает `50` (предварительно на экран выводится цифра 2, т. к. буквенно-цифровой символ "2" имеет код 50, а затем в командную строку выводится возвращаемое значение; поэтому кажется, что возвращено значение 250, а не 50)
- `(setq syn (write-char 40 fo))` возвращает `40` (если прочитать значение переменной `syn`, то оно тоже будет равно 40)

1.11.11. Функция *prompt*

Выводит сообщение на экран (если используется установка системы AutoCAD на двухэкранный вычислительный комплекс, то выводит сообщение на оба экрана). Чтобы избежать этого дублирования, можно пользоваться для вывода сообщений функцией `princ`.

```
(prompt <сообщение>)
```

Аргумент — текстовая строка.

Возвращаемое значение — `nil`.

Пример:

- `(prompt "Новая тема")` возвращает `nil`

1.11.12. Функция *alert*

Позволяет вывести на экран диалоговое окно с сообщением системы AutoCAD.

```
(alert <сообщение>)
```

Аргумент — текстовая строка.

Возвращаемое значение — `nil`.

Максимальная длина строки <сообщение> зависит от многих вещей (в частности, от платформы) и может быть установлена пользователем самостоятельно. В случае превышения допустимой длины AutoCAD обрезает сообщение. При необходимости вывести сообщение в несколько строк нужно использовать знак конца строки ("\n").

Пример:

♦
```
(setq lines (strcat "Я вас любил: любовь еще, быть может,\n"
"В душе моей угасла не совсем;\nНо пусть она вас больше "
"не тревожит;\nЯ не хочу печалить вас ничем."))
(alert lines)
```

Последнее выражение выводит на экран окно, изображенное на рис. 1.11.

Рис. 1.11. Окно, вызываемое функцией `alert`

В данном примере выведены четыре строки, поскольку трижды был использован знак конца строки ("\n"). Выход из диалогового окна, изображенного на рис. 1.11, — щелчок по кнопке **OK**.

1.11.13. Функция *terpri*

Переводит курсор экрана на новую строку (очередное сообщение будет выводиться с начала следующей строки).

```
(terpri)
```

Возвращаемое значение — `nil`.

Фактически является аналогом `(princ "\n")`, но с другим возвращаемым значением.

1.11.14. Функция *vl-directory-files*

Выводит список с именами файлов и папок, расположенных внутри указанной папки.

```
(vl-directory-files [<папка> <образец> <флаг>])
```

Аргументы: <папка> — текстовая строка (имя папки (каталога), из которой нужно распечатать список с именами файлов и папок); <образец> — текстовая строка

(DOS-шаблон для выбора имен файлов); <флаг> — целое число (флаг управления выводом имен папок).

Если аргумент <папка> равен "" или nil или опущен, то в качестве рассматриваемой папки принимается текущий каталог. Если аргумент <образец> опущен или задан равным nil, в качестве его значения принимается "*.*".

Аргумент <флаг> может принимать только три значения:

- -1 — выводить только имена папок;
- 0 — выводить имена и файлов, и папок;
- 1 — выводить только имена файлов.

По умолчанию в качестве значения аргумента <флаг> принимается 0.

Возвращаемое значение — список с именами найденных файлов и папок или nil, если по заданному шаблону и флагу ничего не найдено.

Примеры:

- (vl-directory-files "d:\\AutoCAD 2006" "acad.*" 1) возвращает ("acad.exe" "acad.exe.config")
- (vl-directory-files) возвращает (". " ".." ...) (список оборван) — список файлов и папок текущего каталога (вероятнее всего, папки Мои документы текущего пользователя)

1.11.15. Функция *vl-file-copy*

Копирует содержимое файла-источника в целевой файл. Если целевой файл существует, то либо копирование не выполняется, либо содержимое файла-источника дописывается в конец целевого файла.

```
(vl-file-copy <источник> <цель> [<добавление>])
```

Аргументы: <источник> — текстовая строка (имя файла-источника, т. е. файла, из которого выполняется копирование); <цель> — текстовая строка (имя целевого файла, т. е. файла, в который выполняется копирование); <добавление> — признак, разрешающий добавление информации в конец целевого файла, если он существовал до операции копирования (если аргумент <добавление> задан и не равен nil, то дописывание разрешено; иначе генерируется сообщение об ошибке).

Если в аргументе <источник> имя файла указано без пути, то файл ищется по путям поиска файлов поддержки. То же относится к поиску или созданию файла, указанного в аргументе <цель>.

Возвращаемое значение — целое число (количество байтов) при успешном выполнении копирования. При ошибке возвращается nil.

Причинами ошибки могут быть, например, такие ситуации:

- не существует файл-источник (или в качестве имени файла-источника задано имя папки);
- целевой файл существует, но добавление в него не разрешено;
- сбой при чтении файла-источника;
- сбой при записи (например, целевой файл защищен от записи);
- совпадают имена файла-источника и целевого файла.

Примеры:

- `(vl-file-copy "acad.lsp" "acad1.lsp")` возвращает `nil` (файл-источник не найден)
- `(vl-file-copy "dim.txt" "dimdup.txt")` может вернуть `28`

1.11.16. Функция *vl-file-delete*

Удаляет файл с заданным именем.

```
(vl-file-delete <файл>)
```

Аргумент — текстовая строка с именем файла.

Возвращаемое значение — `T`, если удаление выполнено; `nil`, если удаление не выполнено.

Пример (продолжение второго примера из предыдущего раздела):

- `(vl-file-delete "dimdup.txt")` возвращает `T`

1.11.17. Функция *vl-file-directory-p*

Проверяет, является ли имя, указанное в качестве аргумента, именем папки.

```
(vl-file-directory-p <имя>)
```

Аргумент — текстовая строка.

Возвращаемое значение — `T`, если аргумент указывает имя папки, и `nil` — в остальных случаях.

Если аргумент `<имя>` не содержит полного пути к файлу, то проверяется только текущий каталог (чаще всего — папка Мои документы текущего пользователя).

Примеры:

- `(vl-file-directory-p "Мои рисунки")` возвращает `T`
- `(vl-file-directory-p "Help")` возвращает `nil`

1.11.18. Функция *vl-file-rename*

Переименовывает файл или папку.

```
(vl-file-rename <старое> <новое>)
```

Аргументы — текстовые строки.

Возвращаемое значение — `T`, если переименование выполнено, и `nil`, если переименование невозможно.

Если аргумент `<старое>` не содержит полного пути к файлу, то переименовываемый файл (папка) ищется только в текущем каталоге.

Пример:

- `(vl-file-rename "Help.txt" "Help2.txt")` возвращает `T`, если в рабочем каталоге найдены файл или папка с именем `"Help.txt"` и переименование возможно

1.11.19. Функция *vl-file-size*

Вычисляет размер файла в байтах.

```
(vl-file-size <имя>)
```

Аргумент — текстовая строка.

Возвращаемое значение — целое число (размер файла в байтах). Если указан несуществующий файл или аргумент является именем папки, то возвращается `nil`.

Если аргумент `<имя>` не содержит полного пути к файлу, то анализируемый файл ищется только в текущем каталоге.

Пример:

- `(vl-file-size "dim.txt")` возвращает `28`

1.11.20. Функция *vl-file-systime*

Вычисляет дату и время последней модификации файла.

```
(vl-file-systime <имя>)
```

Аргумент — текстовая строка с именем файла.

Возвращаемое значение — список из восьми целочисленных элементов:

- год;
- месяц;
- день недели;
- день месяца;
- часы;
- минуты;
- секунды;
- миллисекунды.

В качестве дня недели возвращается его порядковый номер (0 — воскресенье, 1 — понедельник, 2 — вторник и т. д.).

Пример:

- `(vl-file-systime "dim.txt")` может вернуть `(2006 1 3 4 21 3 55 421)`, что означает 4 января 2006 года, 3-й день недели (среда), 21 час 3 мин. 55,421 сек.

1.11.21. Функция *vl-filename-base*

Выделяет имя файла, без расширения и пути.

```
(vl-filename-base <имя>)
```

Аргумент — текстовая строка с полным или кратким именем файла.

Возвращаемое значение — текстовая строка с именем файла (без расширения и пути).

Функция не проверяет существования файла.

Пример:

- `(vl-filename-base "d:\\Cartoon\\Change\\v502.prn")` возвращает `"v502"`

1.11.22. Функция *vl-filename-directory*

Выделяет путь из полного имени файла.

```
(vl-filename-directory <имя>)
```

Аргумент — текстовая строка с именем файла.

Возвращаемое значение — строка (путь к файлу).

Функция не проверяет существования файла. Удобно использовать ее в паре с функцией `findfile`, возвращающей полное имя файла.

Примеры:

- `(vl-filename-directory "d:\\Cartoon\\Change\\v502.prn")` возвращает `"d:\\Cartoon\\Change"`
- `(vl-filename-directory "make.exe")` возвращает `""`

1.11.23. Функция *vl-filename-extension*

Выделяет расширение из имени файла.

```
(vl-filename-extension <имя>)
```

Аргумент — текстовая строка с именем файла.

Возвращаемое значение — текстовая строка с расширением файла, начинается с точки. Если имя файла не содержит расширения, то возвращается `nil`.

Примеры:

- (vl-filename-extension "d:\\Cartoon\\Change\\v502.prn") возвращает ".prn"
- (vl-filename-extension "acad.lsp") возвращает ".lsp"
- (vl-filename-extension "acad") возвращает nil

1.11.24. Функция *vl-filename-mktemp*

Вычисляет уникальное (т. е. не совпадающее с именами существующих файлов) имя для временного файла.

(vl-filename-mktemp [<образец> [<папка> [<расширение>]]])

Аргументы: <образец> — образец префикса, используемого при построении имени временного файла; <папка> — папка, в которой нужно разместить временный файл; <расширение> — расширение, с которым нужно сформировать имя временного файла. Все аргументы — текстовые строки.

Возвращаемое значение — строка с уникальным именем временного файла.

Если аргумент <образец> не задан, то по умолчанию принимается "$VL~~". Если <образец> задает главную часть имени временного файла длиной более пяти символов, то используются только первые пять.

Если аргумент <папка> не задан или задан равным nil, то в качестве папки, в которой предполагается размещение временного файла, принимается папка, вычисляемая по следующему алгоритму:

- папка с именем из аргумента <образец>, если он задан с путем;
- папка, имя которой задано в переменной окружения TMP;
- папка, имя которой задано в переменной окружения TEMP;
- текущий каталог.

Если аргумент <расширение> не задан, то расширение временного файла извлекается из аргумента <образец> (может оказаться и пустой строкой). Если аргумент <расширение> указан без начальной точки, то его значение добавляется к главной части имени временного файла.

При формировании имени временного файла к главной части имени, извлекаемой из аргумента <образец>, добавляется трехзначный номер, обеспечивающий требуемую уникальность имени.

Примеры:

- (vl-filename-mktemp) возвращает "C:\\DOCUME~1\\ … \\LOCALS~1\\Temp\\$VL~~001"
- (vl-filename-mktemp "abcdefgh") возвращает "C:\\DOCUME~1\\ … \\LOCALS~1 \\Temp\\abcde002"

- `(vl-filename-mktemp "xxx" "c:\\Temp" "$xx")` возвращает `"c:\\Temp\\xxx003$xx"`
- `(vl-filename-mktemp "xxx" "c:\\Temp" ".$xx")` возвращает `"c:\\Temp\\xxx004.$xx"`

Замечание

Все файлы с именами, сгенерированными функцией `vl-filename-mktemp` во время сеанса работы в среде Visual LISP, автоматически удаляются при выходе из Visual LISP.

1.11.25. Функция *vl-mkdir*

Создает папку с заданным именем.

```
(vl-mkdir <имя>)
```

Тип аргумента — строка.

Возвращаемое значение — `T` при успешном завершении и `nil` — при ошибке создания.

Примеры:

- `(vl-mkdir "ABC")` возвращает `T` (создана папка ABC внутри текущего каталога)
- `(vl-mkdir "d:\\Temp14")` возвращает `T`
- `(vl-mkdir "d:\\XYZ\\Temp14")` возвращает `nil`, если папка d:\XYZ не существует

1.11.26. Функция *vl-princ-to-string*

Преобразует в строку любой аргумент, по аналогии с тем, как функция `princ` выводит данные в файл.

```
(vl-princ-to-string <аргумент>)
```

Тип аргумента — любой.

Возвращаемое значение — текстовая строка.

Примеры:

- `(vl-princ-to-string 41)` возвращает `"41"`
- `(vl-princ-to-string 'cons)` возвращает `"cons"`
- `(vl-princ-to-string "Tree")` возвращает `"Tree"`
- `(vl-princ-to-string '(12 7 11))` возвращает `"(12 7 11)"`
- `(vl-princ-to-string '(0 . "LINE"))` возвращает `"(0 . LINE)"`
- `(vl-princ-to-string 5.1234567)` возвращает `"5.12346"` (оставлено шесть значащих цифр)

1.11.27. Функция *vl-prin1-to-string*

Преобразует в строку любой аргумент, по аналогии с тем, как функция `prin1` выводит данные в файл.

```
(vl-prin1-to-string <аргумент>)
```

Тип аргумента — любой.

Возвращаемое значение — текстовая строка

Примеры:

- `(vl-prin1-to-string 41)` возвращает `"41"`
- `(vl-prin1-to-string 'cons)` возвращает `"cons"`
- `(vl-prin1-to-string "Tree")` возвращает `"\"Tree\""`
- `(vl-prin1-to-string '(12 7 11))` возвращает `"(12 7 11)"`
- `(vl-prin1-to-string '(0 . "LINE"))` возвращает `"(0 . \"LINE\")"`
- `(vl-prin1-to-string 5.1234567)` возвращает `"5.12346"` (сохранено только шесть значащих цифр)

1.11.28. Пример работы с файлами

Приведенные в предыдущих разделах функции дают пользователю возможность работы только с файлами последовательного доступа. Указателем позиции в файле, с которой далее будет происходить чтение (запись), является переменная, хранящая дескриптор этого файла. Механизм работы с файлами отображен в примере, приведенном в листинге 1.12.

Листинг 1.12. Пример использования функций работы с файлами

```
; Глава 01\Листинг 1.12.lsp
; Операции записи
(setq ff (open "d:\\temp.txt" "w"))
(write-line "Первая строка" ff)
(write-char 65 ff)
(write-char 67 ff)
(write-line "Вторая строка" ff)
(close ff)
; Операции чтения
(setq ff (open "d:\\temp.txt" "r"))
(setq s1 (read-char ff))
(setq s2 (read-char ff))
(setq str1 (read-line ff))
(setq str2 (read-line ff))
(setq str3 (read-line ff))
(close ff)
(setq ff nil)
```

Первое выражение — `(setq ff (open "d:\\temp.txt" "w"))` — создает переменную `ff`, в которую заносится дескриптор файла c:\test.txt, открываемого для записи (в первом аргументе функции `open` обратная косая черта удваивается). Затем выражение `(write-line "Первая строка" ff)` заносит в открытый файл первую запись (текст `"Первая строка"` из 13 символов) и символы с кодами 13 и 10, которые в файлах последовательного доступа разделяют записи.

После этого указатель файла показывает на начало второй записи, куда поочередно — с помощью двукратного применения функции `write-char` — заносятся символы с кодами `65` (буква `A`) и `67` (буква `C`). Затем в продолжение второй записи файла с помощью функции `write-line` заносится текст `"Вторая строка"` (13 символов) и добавляются символы с кодами 13 и 10, означающие конец второй записи файла. Вслед за этим файл с дескриптором `ff` закрывается функцией `close`.

Вторая часть листинга начинается функцией открытия того же файла, но уже в режиме чтения. Затем с помощью двукратного применения функций `read-char` читаются два первых символа первой записи, которые запоминаются в переменных `s1` и `s2`. После этого в переменную `str1` читается остаток первой записи. Далее в переменную `str2` читается вторая запись файла. Третья операция чтения с помощью функции `read-line` приносит `nil`, поскольку в файле d:\temp.txt третьей записи нет. По окончании выполнения рассмотренной программы переменная `ff` очищается (в ней хранился дескриптор файла). Другие переменные получат такие значения:

`s1 = 207` (код символа `П`)

`s2 = 229` (код символа `е`)

`str1 = "рвая строка"`

`str2 = "ACВторая строка"`

`str3 = nil`

Функции `princ`, `prin1`, `print` и `prompt` чаще всего применяются для вывода сообщений в командную строку системы AutoCAD, причем `prin1` и `print` выводят сообщения в кавычках (другие особенности описаны выше). Вместо функции `write-line` для записи в файл можно пользоваться функцией `princ`, но заносить признак конца записи, который выглядит как `"\n"`, пользователь должен сам. Например:

```
(princ "Первая запись\n" ff)
```

1.12. Функции доступа к примитивам AutoCAD

Рисунок в системе AutoCAD имеет организацию, аналогичную организации базы данных, в которой элементы (графические примитивы и неграфические объекты)

имеют списковую структуру. Каждый примитив обладает своим типом. Перечислим все эти типы примитивов в алфавитном порядке:

3DFACE
3DSOLID
ACAD_PROXY_ENTITY
ACAD_TABLE
ARC
ARCALIGNEDTEXT
ARC_DIMENSION
ATTDEF
ATTRIB
BODY
CIRCLE
DIMENSION
ELLIPSE
HATCH
IMAGE
INSERT
LARGE_RADIAL_DIMENSION
LEADER
LINE
LWPOLYLINE
MLINE
MTEXT
OLE2FRAME
POINT
POLYLINE
RAY
REGION
RTEXT
SEQEND
SHAPE
SOLID
SPLINE
TEXT
TOLERANCE
TRACE
VERTEX
VIEWPORT
WIPEOUT
XLINE

Чаще всего наименование типа совпадает с английским именем команды системы AutoCAD, которая создает графический объект.

Функции, рассматриваемые в данном разделе, работают с примитивами рисунка, обращаясь непосредственно к их внутренней структуре. LISP имеет средства выбора графических объектов как по их порядковому номеру, так и по определенным признакам (цвету, слою и т. п.).

1.12.1. Функция *entlast*

Извлекает последний неудаленный основной примитив рисунка. Синтаксис функции:

```
(entlast)
```

Функция возвращает `nil`, если в рисунке нет неудаленных примитивов (например, когда вы только что создали новый рисунок). Иначе возвращается послед-

ний основной примитив в следующем виде: `<Entity name: 7ef8bf68>` (`<Имя объекта: 7ef8bf68>`).

Фактически полученное имя примитива определяет адрес в базе данных рисунка, по которому начинается информация о примитиве.

Полученное значение для дальнейшего использования следует сохранить в переменной, например:

```
(setq eela (entlast))
```

1.12.2. Функция *entnext*

Вычисляет следующий примитив (подпримитив).

```
(entnext [<примитив>])
```

В качестве единственного аргумента функции `entnext` может выступать ранее полученный примитив текущего рисунка. Если функция вызывается без аргумента, то она возвращает первый неудаленный примитив в базе рисунка.

При наличии аргумента функция возвращает следующий примитив либо `nil`, если в качестве аргумента указан последний примитив рисунка (т. е. последний графический объект базы рисунка).

Пример:

- `(setq e1 (entnext))` возвращает первый неудаленный примитив рисунка
- `(setq e2 (entnext e1))` возвращает примитив, следующий за `e1`
- `(setq e3 (entnext e2))` возвращает примитив, следующий за `e2`

Функция `entlast` возвращает последний основной примитив. Это означает, что если последним созданным графическим объектом рисунка является сложный объект (например, полилиния типа POLYLINE), то вслед за ним в базе рисунка следуют подпримитивы, т. е. вершины (имя примитива — VERTEX), а завершается перечисление подпримитивов полилинии примитивом SEQEND.

Пример, приведенный в листинге 1.13, предполагает, что объект типа POLYLINE (полилиния или сеть) является последним основным примитивом.

Листинг 1.13. Пример доступа к вершинам примитива POLYLINE

```
; Глава 01\Листинг 1.13.lsp
; Выбираем последний примитив
; (в предположении, что им является POLYLINE)
(setq eela (entlast))
(setq v1 (entnext eela))
; v1 - примитив VERTEX, являющийся первой вершиной POLYLINE
(setq v2 (entnext v1))
; v2 - примитив VERTEX, являющийся второй вершиной POLYLINE
```

Замечание

В компактной полилинии (примитиве типа LWPOLYLINE) нет подпримитивов, и информация обо всех вершинах может быть извлечена из основного примитива.

1.12.3. Функция *entsel*

Функция `entsel` предлагает пользователю указать один объект, выдавая запрос, текст которого задан в качестве аргумента.

```
(entsel [<запрос>])
```

Аргумент `<запрос>` — любая строка текста.

Возвращаемое значение — список, состоящий из двух элементов: имени выбранного примитива и точки, которой пользователь указал объект (такая точка, как правило, оказывается вне самого примитива, поскольку точность указания мышью зависит от величины прицела). Можно задать объект вводом на клавиатуре ключевого слова **Last** (Последний) — тогда в возвращаемом списке координаты точки будут нулевыми.

Пример:

- `(setq es1 (entsel "Выберите объект: "))` возвращает `(<Entity name: 7ef66fa0> (475.13 248.759 0.0))`

Указанный пользователем объект фигурирует в качестве первого элемента списка, являющегося возвращаемым значением, и может быть извлечен из такого списка, например, с помощью функции `car`.

1.12.4. Функция *entdel*

Позволяет удалять неудаленные основные примитивы и восстанавливать примитивы, ранее удаленные в данном сеансе редактирования.

```
(entdel <примитив>)
```

Функция возвращает имя удаляемого (восстанавливаемого) примитива. Следует иметь в виду, что при операции сохранения рисунка все примитивы, помеченные как удаленные, из рисунка стираются и уже больше не могут быть восстановлены.

В листинге 1.14 приведен пример удаления и восстановления примитива с помощью функции `entdel`.

Листинг 1.14. Пример использования функции `entdel`

```
; Глава 01\Листинг 1.14.lsp
; Строим отрезок для манипуляций
(command "_.LINE" '(16.0 66.1) '(250.3 145.9) "")
(setq k1 (entdel (entlast)))
```

```
; Только что нарисованный отрезок удален
(entdel k1)
; Удаленный примитив восстановлен
```

1.12.5. Функция *entget*

Получает список с характеристиками примитива и является основным инструментом извлечения информации о геометрии и свойствах.

```
(entget <примитив> [<приложения>])
```

Здесь аргумент `<примитив>` — это имя примитива для получения его данных, аргумент `<приложения>` — это список с именами приложений, с помощью которых к примитиву привязаны расширенные данные *(о расширенных данных см. разд. 1.14)*.

Предположим, в рисунке первым объектом является отрезок (тип примитива — LINE). Тогда выражение

```
(setq le (entget (entnext)))
```

должно вернуть примерно такой список:

```
((-1 . <Entity name: 7ef66fa0>) (0 . "LINE") (330 . <Entity name: 7ef66cf8>)
(5 . "EC") (100 . "AcDbEntity") (67 . 0) (410 . "Model")
(8 . "0") (100 . "AcDbLine") (10 214.31 409.132 0.0)
(11 608.276 172.218 0.0) (210 0.0 0.0 1.0))
```

В этом списке элементами являются точечные пары и списки, причем и в тех и в других первыми элементами выступают целые числа, называемые *DXF-кодами (см. также разд. 1.9.11 и приложение 2)*. Под соответствующим кодом в точечных парах и списках находятся данные определенного типа:

- код –1 указывает имя примитива (`<Entity name: 7ef66fa0>`);
- код 0 — тип примитива (`"LINE"`);
- код 5 — метку (внутренний номер примитива в рисунке);
- код 8 — имя слоя (`"0"`);
- код 10 — координаты начальной точки (`214.31 409.132 0.0`);
- код 11 — координаты конечной точки (`608.276 172.218 0.0`);
- код 210 — направление вектора нормали к плоскости, в которой создан примитив;
- код 410 — имя вкладки пространства модели или листа.

Остальные коды, не имеющие принципиального значения, здесь не рассматриваются. С помощью функции `assoc` можно из списка с характеристиками объекта извлечь нужную точечную пару, а затем функцией `cdr` получить данные необходимого DXF-кода.

Продолжим пример с отрезком:

- `(cdr (assoc 0 le))` возвращает `"LINE"` (тип примитива)
- `(cdr (assoc 8 le))` возвращает `"0"` (имя слоя)
- `(cdr (assoc 10 le))` возвращает `(214.31 409.132 0.0)`
- `(cdr (assoc 11 le))` возвращает `(608.276 172.218 0.0)`

Кроме того, по коду 62 можно было бы извлечь номер цвета[4] примитива, по коду 6 — имя типа линий, по коду 48 — собственный масштаб типа линий, по коду 311 — вес линий (умноженный на 100). В рассмотренном выше списке `le` точечных пар с такими DXF-кодами нет, поэтому для них действуют значения по умолчанию: ByLayer (ПоСлою) или 1.

Рассмотрим, какой примерный список можно было бы получить для окружности (примитива типа CIRCLE):

```
((-1 . <Entity name: 7ef7c008>) (0 . "CIRCLE")
(330 . <Entity name: 7ef66cf8>) (5 . "F1") (100 . "AcDbEntity") (67 . 0)
(410 . "Model") (8 . "0") (100 . "AcDbCircle") (10 1263.06 418.244 0.0)
(40 . 85.6843) (210 0.0 0.0 1.0))
```

Для окружности DXF-коды –1, 0, 410, 8, 210 имеют тот же смысл, что и для всех примитивов (в том числе и для примитива типа LINE). Под кодом 40 находится радиус окружности, а точечная пара с DXF-кодом 10 хранит в себе данные о центре.

Поэтому для рассматриваемой окружности:

- `(cdr (assoc 40 le))` возвращает `85.6843` (радиус)
- `(cdr (assoc 10 le))` возвращает `(1263.06 418.244 0.0)` (точка центра)

Сравнивая справочную информацию, выдаваемую командой LIST (СПИСОК), и список, выводимый функцией `entget`, можно получить геометрический смысл DXF-кодов для примитивов других типов.

1.12.6. Функция *entmake*

Создает новый примитив по списку, структура которого аналогична структуре списка, возвращаемого функцией `entget`. Возможно построение примитива с расширенными данными *(о расширенных данных см. разд. 1.14)*. Функция `entmake` не может создать примитив типа VIEWPORT, но применима к объектам символьных таблиц (слоев, видов и т. п. — *см. разд. 1.13*).

```
(entmake <список>)
```

[4] При использовании полной палитры цветов DXF-код 420 содержит значение цвета, а DXF-код 430 — имя из альбома цветов.

Аргумент <список> — это список, описывающий примитив (кроме точечной пары с DXF-кодом –1). По такому списку можно воссоздать в рисунке примитив с помощью функции entmake.

Возвращаемое значение — аргумент <список> или nil, если создание объекта невозможно.

Пример:

- (entmake '((0 . "CIRCLE") (10 500.0 0.0 0.0) (40 . 50.0))) создает новый примитив — окружность с центром в точке (500 0 0) и радиусом 50; остальные свойства (слой, цвет и т. д.), данные о которых отсутствуют в списке, берутся по умолчанию; возвращаемое значение — ((0 . "CIRCLE") (10 500.0 0.0 0.0) (40 . 50.0))
- (entmake '((0 . "LAYER") (100 . "AcDbSymbolTableRecord") (100 . "AcDbLayerTableRecord") (2 . "MyNew") (70 . 0) (62 . 1) (6 . "Continuous"))) создает новый объект в таблице слоев — слой с наименованием MyNew, цветом 1 и типом линий CONTINUOUS

1.12.7. Функция *entmakex*

Создает новый примитив или неграфический объект по списку с данными, аналогично функции entmake. Является основным инструментом создания словарей и X-записей *(см. разд. 1.15)*.

```
(entmakex <список>)
```

Возвращаемое значение — новый примитив или nil, если создание объекта невозможно.

Пример:

- (entmakex '((0 . "CIRCLE") (10 500.0 0.0 0.0) (40 . 50.0))) создает новый примитив — окружность с центром в точке (500 0 0) и радиусом 50. Возвращается примитив, например: <Entity name: 7ef7c010>

1.12.8. Функция *entmod*

Функция entmod похожа на функцию entmake, но получает в качестве аргумента список, который содержит точечную пару с DXF-кодом –1 (т. е. существующий в рисунке примитив) и модифицирует примитив в соответствии с новым списком (в списке могут изменяться любые данные, кроме имени примитива, типа примитива и метки, т. е. точечных пар с кодами –1, 0 и 5):

```
(entmod <список>)
```

Функция entmod изменяет примитив в базе рисунка и возвращает значение аргумента <список> при успешном завершении или nil — при невозможности выполнить изменение. Данная функция не работает с примитивом VIEWPORT.

В листинге 1.15 на примере отрезка с цветом ByLayer (ПоСлою) показан метод изменения примитива с помощью функции `entmod`.

Листинг 1.15. Пример использования функции `entmod`

```
; Глава 01\Листинг 1.15.lsp
; Выбираем отрезок с цветом ByLayer (ПоСлою)
(setq le (car (entsel)))
; Для отрезка возвращается список примерно такого вида:
; ((-1 . <Entity name: 7ef66fa0>) (0 . "LINE")
; (330 . <Entity name: 7ef66cf8>) (5 . "EC") (100 . "AcDbEntity")
; (67 . 0) (410 . "Model") (8 . "0") (100 . "AcDbLine")
; (10 214.31 409.132 0.0) (11 608.276 172.218 0.0) (210 0.0 0.0 1.0))
; В списке нет точечной пары с DXF-кодом 62
; (это означает, что цвет имеет значение ByLayer (ПоСлою))
(setq le (append le (list (cons 62 1))))
; К списку добавлена точечная пара для красного цвета
(entmod le)
; Цвет отрезка изменился на красный
```

Если бы в данном примере список с данными отрезка уже содержал точечную пару с DXF-кодом 62, то изменение цвета пришлось бы выполнить заменой значения, например:

- `(setq le (subst (cons 62 5) (assoc 62 le) le))` возвращает список, в котором точечная пара цвета заменена на точечную пару с синим цветом (номер 5)

  `(entmod le)` изменяет цвет отрезка на синий

1.12.9. Функция *entupd*

Перерисовывает примитив на экране с учетом изменений, внесенных, например, функцией `entmod`.

```
(entupd <примитив>)
```

Аргумент `<примитив>` — это имя примитива в том виде, в котором оно выводится, например, функцией `entlast`.

Возвращаемое значение — имя примитива или `nil` (при ошибке).

Пример:

- `(entupd (entlast))` перерисовывает последний примитив

1.12.10. Функция *handent*

Получает имя примитива или неграфического объекта по его метке.

```
(handent <метка>)
```

Аргумент <метка> — это текстовая строка с шестнадцатеричной меткой в том виде, в каком она возвращается функцией entget.

Возвращаемое значение — имя примитива или nil (при ошибке).

Пример:

- (handent (cdr (assoc 5 le))) возвращает примитив по данным его метки (le — список с данными примитива)

1.12.11. Функция *nentsel*

Запрашивает объект и для простого примитива возвращает такой же список из имени примитива и точки указания, как и функция entsel. Однако если указанный пользователем примитив является трехмерной полилинией (POLYLINE), то первым элементом возвращаемого списка будет имя подпримитива начальной вершины (VERTEX) участка, на котором указывалась полилиния. Если указанный пользователем объект является вхождением блока (INSERT), то возвращается список из двух или четырех элементов (см. ниже).

```
(nentsel [<запрос>])
```

Аргумент <запрос> — строка. Если аргумент не задан, то в качестве подсказки выводится стандартный запрос: ***Select object:*** *(Выберите объект:).*

Функция nentsel работает с блоками (примитивами типа INSERT) следующим образом. В случае если пользователь указал атрибут блока, то функция возвращает список из двух элементов: первым является имя объекта-атрибута, а вторым — точка указания атрибута. В случае если пользователь указал не атрибут, а объект, принадлежащий вхождению блока, то возвращается список из четырех элементов: первым является примитив, с помощью которого был указан блок, вторым — точка указания, третьим — матрица размером 4×3 для преобразования точек из системы координат объекта (СКО) в МСК, четвертым — список с именем блока, содержащим указанный примитив (если примитив входит в блок, который вложен в другой блок, то список содержит все имена вкладываемых блоков, начиная от самого внутреннего и кончая самым внешним).

Матрица размером 4×3, которая выдается в качестве третьего элемента возвращаемого функцией nentsel значения, имеет вид: ((m00 m01 m02) (m10 m11 m12) (m20 m21 m22) (m30 m31 m32)). Тогда преобразование точек из СКО в МСК идет по системе уравнений:

```
X' = X*m00 + Y*m10 + Z*m20 + m30
Y' = X*m01 + Y*m11 + Z*m21 + m31
Z' = X*m02 + Y*m12 + Z*m22 + m32
```

Здесь (X Y Z) — координаты точки до преобразования, (X' Y' Z') — координаты точки после преобразования.

Пример:

- `(nentsel "Укажи блок: ")` возвращает для блока `(<Entity name: 7ef7c068> (1179.16 400.02 0.0) ((1.0 0.0 0.0) (0.0 1.0 0.0) (0.0 0.0 1.0) (1230.23 403.664 0.0)) (<Entity name: 7ef7c070>))`

1.12.12. Функция *nentselp*

Функция `nentselp` аналогична функции `nentsel`, но если указанным примитивом является вхождение блока (INSERT), то возвращает список, в котором третьим элементом является матрица преобразования размером не 4×3, а 4×4 (см. ниже).

```
(nentselp [<запрос>] [<точка>])
```

Если задан аргумент `<точка>`, то запрос не выдается и аргумент выступает в качестве точки указания.

Функция `nentselp` работает аналогично функции `nentsel`, но в случае если пользователь указал объект, принадлежащий вхождению блока, то, как и `nentsel`, тоже возвращается список из четырех элементов, однако третьим элементом в списке является матрица размером 4×4, которая служит для преобразования точек из системы координат объекта (СКО) в МСК. Матрица имеет вид: `((n00 n01 n02 n03) (n10 n11 n12 n13) (n20 n21 n22 n23) (0.0 0.0 0.0 1.0))`. Преобразование точек из СКО в МСК идет по такой системе уравнений:

```
X' = X*n00 + Y*n01 + Z*n02 + n03
Y' = X*n10 + Y*n11 + Z*n12 + n13
Z' = X*n20 + Y*n21 + Z*n22 + n33
```

Смысл списков `(X Y Z)` и `(X' Y' Z')` тот же, что и для функции `nentsel`.

Пример:

- `(nentselp "Укажи блок: ")` возвращает для блока `(<Entity name: 7ef7c068> (1180.99 445.58 0.0) ((1.0 0.0 0.0 1230.23) (0.0 1.0 0.0 403.664) (0.0 0.0 1.0 0.0) (0.0 0.0 0.0 1.0)) (<Entity name: 7ef7c070>))`

1.12.13. Наборы

Следующая группа функций — это функции, работающие с наборами объектов. *Набор* — это временное множество, в которое входят имена основных примитивов рисунка (они собираются по какому-то условию или признаку-фильтру). Неосновные примитивы (составные части блока или вершины полилинии) не могут входить в набор.

Имя набора обычно сохраняется в какой-нибудь переменной, чтобы затем можно было извлечь из набора нужные примитивы для анализа или операций редактирования.

Наборы уничтожаются при закрытии рисунка.

Замечание

При необходимости сохранять наборы примитивов в рисунке следует оформлять их в виде групп с помощью команды GROUP (ГРУППА). Группы относятся к неграфическим объектам, существующим внутри рисунка.

Количество одновременно открытых наборов в рисунке не должно превышать 128. В ранних версиях системы AutoCAD количество наборов, которые могут быть одновременно открыты, было меньше (например, в 10-й версии оно равнялось всего шести).

Для того чтобы закрыть ненужный набор, переменной, в которой хранится имя этого набора, следует присвоить значение `nil`.

1.12.14. Функция *ssget*

Функция `ssget` формирует набор по запросу или признаку.

```
(ssget [<метод>] [<точка1> [<точка2>]] [<список>] [<фильтр>])
```

Аргумент `<метод>` — это текстовая строка, принимающая одно из значений, характеризующих метод выбора примитивов (для возможности использования и в локализованной версии системы AutoCAD все перечисляемые строки должны спереди дополняться символом подчеркивания).

К основным значениям относятся следующие:

- `"X"` — выбор всех примитивов рисунка, включая объекты на замороженных слоях и вне видимой части экрана, с учетом возможного аргумента `<фильтр>`;
- `"A"` — выбор всех примитивов рисунка, включая объекты на замороженных слоях и вне видимой части экрана;
- `"I"` — текущий набор из объектов, отмеченных с помощью ручек;
- `"P"` — последний сформированный ранее набор;
- `"W"` — выбор рамкой;
- `"WP"` — выбор многоугольником;
- `"C"` — выбор секущей рамкой;
- `"CP"` — выбор секущим многоугольником;
- `"F"` — выбор секущей линией;
- `"L"` — выбор последнего видимого примитива.

Дополнительные значения (как правило, являются установками каких-то особенностей выбора):

- `":D"` — разрешение на повторное включение примитивов в набор;
- `":A"` — отмена разрешения на повторное включение примитивов в набор;

- `":E"` — разрешение на выбор всех примитивов, попадающих в прицел устройства указания;
- `":L"` — запрещение выбора примитивов на блокированных слоях;
- `":N"` — разрешение на выбор основных примитивов с помощью подпримитивов (сегментов подробной полилинии или сети[5], компонентов блока); используется только при графическом варианте выбора и только для указания рамкой, секущей рамкой и точкой; может привести к повторению примитивов в наборе, поскольку, например, можно отметить одну и ту же подробную полилинию с помощью любого из ее участков *(см. также функцию `ssnamex`)*;
- `":S"` — допускает только один интерактивный выбор (по умолчанию возможен многошаговый выбор примитивов).

Функция располагает и другими недокументированными возможностями, наследуемыми от соответствующих функций ObjectARX (например, `"M"`, `"."`, `"#"`, `":O"`, `":?"`,`":$"`, комбинирование режимов типа `":D:E"` и т. д.).

Аргумент `<точка1>` при отсутствии аргумента `<точка2>` определяет дополнительное условие для добавления в набор: включаемый примитив должен проходить через точку, заданную аргументом `<точка1>`. Если заданы оба аргумента `<точка1>` и `<точка2>`, то определяемые ими точки являются углами простой или секущей рамки (когда в качестве метода выбора применяются рамки, т. е. `"W"` или `"C"`).

Аргумент `<список>` представляет собой список, элементами которого являются точки, используемые в методах `"F"`, `"WP"`, `"CP"`.

Аргумент `<фильтр>` — это список со структурой, аналогичной структуре списка, возвращаемого функцией `entget`, что позволяет оставить только те примитивы, которые удовлетворяют дополнительным признакам (например, определенному цвету, слою, весу и т. д.).

Возвращаемое значение функции `ssget` — имя созданного набора (или `nil`, если создать набор невозможно). Имя набора возвращается в следующем виде:

```
<Selection set: 2>
```

Фактически имя набора — это адрес в базе данных рисунка, по которому начинается информация об объектах, включенных в набор.

Примеры:

- `(ssget)` выдает стандартный запрос ***Select objects:*** *(Выберите объекты:)* и затем создает набор в соответствии с интерактивными действиями пользователя
- `(ssget "_:S")` аналогичен предыдущему варианту, но разрешает сделать только один выбор

---

[5] Примитивы POLYLINE.

- `(ssget '(125.4 58.1))` создает набор из всех примитивов, проходящих через точку `(125.4 58.1)`
- `(ssget "_X" '(125.4 58.1))` выдает сообщение об ошибке (несовместимость режимов)
- `(ssget '(125.4 58.1) '((0 . "POLYLINE")))` создает набор из всех примитивов типа POLYLINE, проходящих через точку `(125.4 58.1)`
- `(ssget "_C" '(14.3 -5.2) '(27.6 106.1))` создает набор из примитивов, выбираемых секущей рамкой с углами в точках `(14.3 -5.2)` и `(27.6 106.1)`
- `(ssget "_F" '((1.0 0.0) (2.0 1.0) (10.0 1.0) (12.0 0.0)))` создает набор из примитивов, пересекаемых ломаной линией, имеющей четыре вершины
- `(ssget "_X" (list (cons 0 "LWPOLYLINE") (cons 8 "SET")))` создает набор из всех примитивов типа LWPOLYLINE, размещенных на слое SET
- `(ssget "_X" '((0 . "LWPOLYLINE") (8 . "SET,ACR,BELL")))` создает набор из всех примитивов типа LWPOLYLINE, размещенных на слоях SET, ACR и BELL
- `(ssget "_X" '((0 . "*LINE")))` создает набор из всех примитивов рисунка, являющихся объектами типа *LINE (т. е. LINE, SPLINE, POLYLINE, LWPOLYLINE)
- `(ssget "_X" '((8 . "СТЕНА[1-3]")))` создает набор из всех примитивов рисунка на слоях СТЕНА1, СТЕНА2, СТЕНА3
- `(ssget "_A")` создает набор из всех примитивов рисунка, аналогично методу `"_X"` с пустым фильтром, т. е. `(ssget "_X")`, `(ssget "_X" '())` или `(ssget "_X" nil)`

В рисунке одновременно может быть открыто не более 128 наборов. При достижении такого предела функция `ssget` отказывается создавать следующие наборы и возвращает для них `nil`. Чтобы удалить ненужные наборы, следует присвоить значение `nil` переменным, хранящим имена этих наборов.

Из примеров видно, что в фильтрах со строковыми значениями можно использовать те же возможности, что и в функции `wcmatch` (т. е. применять символы *, #, @, ?, [,] и т. д.).

1.12.14.1. Фильтры со сложными условиями

Для формирования **сложных условий** в фильтрах выбора применяются точечные пары со служебным DXF-кодом –4. В них используются следующие аналоги логических операторов: AND, OR, XOR, NOT. Каждый из этих операторов имеет два вида: открывающей скобки и закрывающей скобки, например: "<AND" и "AND>" (верхний регистр не обязателен). Между такими скобками размещаются проверяемые значения (AND и OR являются многоместными операторами, XOR — двухместным, NOT — одноместным):

- `(ssget "_X" '((-4 . "<OR") (8 . "0") (0 . "LINE") (-4 . "OR>")))` создает набор из тех примитивов, которые или расположены на слое 0, или являются отрезками (примитивами LINE)
- `(ssget "_X" '((-4 . "<OR") (-4 . "<AND") (8 . "0") (0 . "CIRCLE") (-4 . "AND>") (0 . "LINE") (-4 . "OR>")))` создает набор из тех примитивов, которые или являются окружностями и расположены на слое 0, или являются отрезками
- `(ssget "_X" (list (cons 0 "LINE") (cons -4 "<NOT") '(8 . "0") (cons -4 "NOT>")))` создает набор из тех примитивов, которые являются отрезками, но не расположены на слое 0

В точечных парах с числовыми значениями можно использовать логические операторы, приведенные в табл. 1.7.

Таблица 1.7. *Логические операторы, используемые в фильтрах выбора*

Оператор	Пояснение	Оператор	Пояснение
*	Любое (не проверять)	<=	Меньше или равно
=	Равно	>	Больше
!=, /=	Не равно	>=	Больше или равно
<>	Не равно	&	Побитовое И (для целых)
<	Меньше	&=	Побитовое И с равенством (для целых)

Операторы & и &= работают только с целыми числами. Оператор & возвращает истинное значение, если хотя бы один из битов, установленных в маске, установлен и в проверяемом числе. Оператор же &= возвращает истинное значение только тогда, когда все биты, установленные в маске, установлены и в проверяемом числе.

Все остальные операторы, приведенные в табл. 1.7, применимы как к целым, так и к вещественным значениям. Примеры:

- `(ssget "_X" (list (cons 0 "CIRCLE") '(-4 . ">=") '(370 . 100)))` создает набор из окружностей, которые имеют значение веса, большее или равное 1,00 мм (напомним, что в DXF-коде 370 веса хранятся числами в 100 раз больше, т. е. 100 соответствует 1,00 мм)
- `(ssget "_X" '((0 . "CIRCLE") (-4 . ">=") (370 . 100) (-4 . "<") (40 . 225.0)))` создает набор из окружностей, которые имеют значение веса, большее или равное 1,00 мм, и радиус меньше 225 мм
- `(ssget "_X" '((0 . "LWPOLYLINE") (-4 . "&") (70 . 1)))` создает набор из замкнутых компактных полилиний (проверяется флаг на установку бита замкнутости)

- `(ssget "_X" '((0 . "POLYLINE") (-4 . "&") (70 . 6)))` создает набор из примитивов POLYLINE, сглаженных любой из двух опций команды PEDIT (ПОЛРЕД): **Fit** (СГладить) и **Spline** (СПлайн)
- `(ssget "_X" '((0 . "POLYLINE") (-4 . "&=") (70 . 6)))` возвращает `nil`, т. к. полилиния не может быть одновременно сглажена и вариантом **Fit** (СГладить), и вариантом **Spline** (СПлайн)
- `(ssget "_X" '((0 . "CIRCLE") (-4 . ">,>,*") (10 100.0 230.0 99.0)))` создает набор из окружностей, точки центров которых удовлетворяют таким условиям: $X > 100$, $Y > 230$, Z любые (обратите внимание на комбинацию `">,>,*"`, используемую для трехкоординатного DXF-кода)

При проверке значения DXF-кода 210 (вектор нормали) применимы только операторы *, =, !=, /=, <>.

1.12.15. Функция *ssadd*

Добавляет примитив в набор.

```
(ssadd [<примитив> [<набор>]])
```

Аргументы: `<примитив>` — примитив, `<набор>` — набор.

Возвращаемое значение — набор.

Если аргумент `<набор>` опущен, а аргумент `<примитив>` задан, то создается новый набор из одного примитива. Если оба аргумента опущены, то создается новый пустой набор (но его значение не равно `nil`).

Пример работы с функцией `ssadd` приведен в листинге 1.16.

Листинг 1.16. Пример использования функции `ssadd`

```
; Глава 01\Листинг 1.16.lsp
; Строим вспомогательный отрезок
(command "_.LINE" '(14.0 79.2) '(35.1 81.3) "")
(setq ss1 (ssadd)); создает пустой набор
(ssadd (entlast) ss1)
; В набор ss1 добавлен последний примитив (типа LINE)
```

1.12.16. Функция *ssdel*

Удаляет примитив из набора.

```
(ssdel <примитив> <набор>)
```

Аргументы: `<примитив>` — удаляемый примитив, `<набор>` — набор, из которого нужно выполнить удаление.

Возвращаемое значение — набор, если операция удаления выполнена, или `nil`, если примитив не содержится в наборе и удалить его нельзя.

Пример (продолжение примера из *разд. 1.12.15*):

- `(ssdel (entlast) ss1)` удаляет из набора `ss1` последний примитив (типа LINE) и возвращает `<Selection set: 4>`

1.12.17. Функция *sslength*

Вычисляет длину набора, т. е. количество содержащихся в нем примитивов.

```
(sslength <набор>)
```

Аргумент `<набор>` — набор.

Возвращаемое значение — целое число.

Примеры:

- `(sslength (ssadd))` возвращает 0, т. к. функция `ssadd` создала пустой набор
- `(sslength (ssget "_L"))` возвращает 1, т. к. функция `ssget` создала набор из одного примитива (последнего видимого примитива рисунка)

1.12.18. Функция *ssmemb*

Проверяет, входит ли примитив в набор.

```
(ssmemb <примитив> <набор>)
```

Аргументы: `<примитив>` — примитив, `<набор>` — набор.

Возвращается примитив (значение аргумента `<примитив>`), если он входит в рассматриваемый набор, или `nil` — если не входит.

Примеры:

- `(setq nabor (ssget "_L"))` создает набор из последнего примитива и возвращает `<Selection set: 7>`
- `(ssmemb (entlast) nabor)` возвращает примитив (например, `<Entity name: 7ef7c010>`), т. к. последний примитив содержится в наборе `nabor`

1.12.19. Функция *ssname*

Извлекает примитив из набора по порядковому номеру.

```
(ssname <набор> <номер>)
```

Аргумент `<набор>` — набор. Аргумент `<номер>` определяет номер элемента в наборе. Значение должно быть либо целым числом от 0 до 32767, либо целым числом в вещественной форме больше 32767.0. Нумерация примитивов в наборе начинается с нуля.

Возвращаемое значение — примитив или nil, если примитива с данным номером в наборе нет.

Примеры (продолжение примера из *разд. 1.12.18*):

- (ssname nabor 0) возвращает имя примитива (например, <Entity name: 7ef7c010>)
- (ssname nabor 1) возвращает nil, т. к. в наборе nabor содержится только один объект

См. также примеры в *разд. 1.12.23*.

1.12.20. Функция *ssnamex*

Извлекает примитив из набора по порядковому номеру (заданных соответствующими аргументами <набор> и <номер>), но при этом выдается дополнительная информация о методах формирования набора (см. ниже). Если аргумент <номер> опущен, то выводится информация обо всех примитивах набора.

```
(ssnamex <набор> [<номер>])
```

Типы аргументов — как у функции ssname.

Функция ssnamex возвращает список, в котором элементами являются описания способа выбора примитивов, попавших в набор. Если аргумент <номер> не задан, то длина возвращаемого списка равна количеству элементов, определяемых аргументом <набор>. Если аргумент <номер> задан, причем корректно, то возвращаемым значением будет список из одного элемента — описания способа выбора примитива с порядковым номером, равным значению аргумента <номер>. Если же аргумент <номер> задан неправильно, то функция ssnamex вернет nil.

Описание способа выбора примитива — это список из трех элементов: (<метод> <примитив> <данные>). Параметр <метод> — целое число, которое соответствует методу выбора функции ssget и может принимать такие значения:

- 0 — неинтерактивный выбор ("L", "A", "X" и т. п.);
- 1 — выбор указанием точки;
- 2 — "W" или "WP";
- 3 — "C" или "CP";
- 4 — "F".

Если на месте параметра <метод> стоит отрицательное число, то оно определяет многоугольник выбора (см. описание многоугольника ниже).

Параметр <примитив> описания способа выбора примитива является его именем в том виде, в котором оно выводится функцией entlast. Параметр <данные> может быть либо только числом 0, либо числом 0 и списком, уточняющим выбор, либо числом 0 и номером многоугольника. В уточняющем списке может быть точка,

с помощью которой выбран объект. Если использован номер многоугольника (а он должен быть отрицательным: –1, –2 и т. п.), то далее будет следовать описание многоугольника с соответствующим номером.

Описание многоугольника — это список такого вида: `(<номер_многоугольника> <точка1> ... <точкаN>)`. Нумерация многоугольников начинается с –1 и изменяется путем добавления –1 к очередному номеру.

Описание точки (`<точка1>` и т. п.) — это список из трех (или двух) элементов: `(<код> <база> [<вектор>])`. Описание точки зависит от вида, на котором она выбиралась, и фактически представляет собой описание прямой, луча или отрезка. Параметр `<код>` является кодом описания точки (0 — прямая, 1 — луч, 2 — отрезок), параметр `<база>` представляет собой начальную точку описания, а необязательный параметр `<вектор>` — это либо направление, в котором продолжается бесконечная линия (прямая или луч), либо смещение в сторону, характерную для отрезка.

Например, если в набор `nab5` включен один объект, который был указан точкой `(7.51 16.02)` на обычном виде сверху в МСК, то выражение `(ssnamex nab5 0)` возвращает нечто похожее на `((1 <Entity name: 7ef7da10> 0 (0 (7.51 16.02 0.0))))`.

В этом примере возвращенный функцией `ssnamex` список состоит из одного элемента (списка), в котором присутствуют следующие составные части:

- `<метод>` — равен `1` (соответствует выбору указанием точки);
- `<примитив>` — `<Entity name: 7ef7da10>`;
- `<данные>` — `0 (0 (7.51 16.02 0.0))`, что соответствует точке `(7.51 16.02 0.0)`, примененной для выбора примитива.

Другие примеры значений, возвращаемых функцией `ssnamex`:

- `((0 <Entity name: 7ef7da10> 0))` — один примитив, выбранный неинтерактивным методом (например, "L");
- `((3 <Entity name: 14ae568> 0 -1) (3 <Entity name: 14ae578> 0 -1) (-1 (0 (-147.787 129.133 0.0)) (0 (-64.2728 129.133 0.0)) (0 (-64.2728 22.3376 0.0)) (0 (-147.787 22.3376 0.0))))` — два примитива, выбранных секущим прямоугольником с номером –1 (для прямоугольника заданы четыре точки);
- `((1 <Entity name: 14ae560> 0 (0 (74.2672 64.8218 0.0))) (2 <Entity name: 14ae570> 0 -1) (2 <Entity name: 14ae568> 0 -1) (-1 (0 (-177.446 158.755 0.0)) (0 (40.3151 158.755 0.0)) (0 (40.3151 83.1407 0.0)) (0 (-177.446 83.1407 0.0))))` — три примитива, первый из которых выбран указанием с помощью точки `(74.2672 64.8218 0.0)`, а второй и третий выбраны рамкой, которая фигурирует как многоугольник с номером –1.

В этих примерах параметр `<вектор>` описания точек нигде не использован, т. е. объекты выбирались на виде сверху в МСК.

Наборы — удобный инструмент для выбора нужных примитивов рисунка с заданными свойствами. После формирования набора с помощью функции ssname (или ssnamex) из него извлекаются объекты, с которыми далее выполняются необходимые операции (например: удаление, модификация и т. д.).

1.12.21. Функция *sssetfirst*

Включает ручки у примитивов, входящих в наборы, заданные аргументами <набор1> и <набор2>, и подсвечивает пунктиром примитивы из набора, заданного аргументом <набор2> (что означает выбор их для следующей операции редактирования).

```
(sssetfirst <набор1> [<набор2>])
```

Аргументы — наборы или nil. В качестве первого аргумента в версиях 2004—2006 рекомендуется давать nil (см. примечание далее).

Возвращаемое значение — список с двумя наборами, переданными в качестве аргументов.

Пример использования функции sssetfirst приведен в листинге 1.17.

Листинг 1.17. Пример использования функции sssetfirst

```
; Глава 01\Листинг 1.17.lsp
; Строим вспомогательный отрезок
(command "_.LINE" '(276.5 42.0) '(90.3 71.8) "")
(setq ss1 (ssadd (entlast)))
; Создан набор ss1 из одного отрезка
; Строим вспомогательную окружность
(command "_.CIRCLE" '(180.0 93.0) 72.0)
(setq ss2 (ssadd (entlast)))
; Создан набор ss2 из одной окружности
(sssetfirst ss1 ss2); возвращает (<Selection set: 6> <Selection set: 8>)
; Включает ручки и подсвечивает окружность
(sssetfirst nil ss2); тот же результат
```

Результат выполнения примера показан на рис. 1.12.

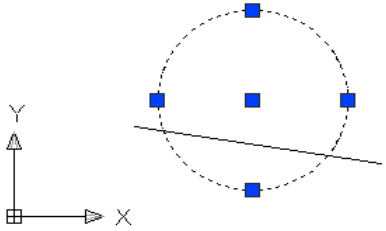


Рис. 1.12. Результат выполнения функции sssetfirst

Примечание

До версии 2002 в этом примере AutoCAD включал ручки и у отрезка (без подсветки). Но начиная с версии 2004 с первым набором ничего не происходит. Поэтому имеет смысл в качестве первого аргумента функции `sssetfirst` использовать только `nil`, если другого не требует последующий вызов функции `ssgetfirst`.

1.12.22. Функция *ssgetfirst*

Возвращает список с именами двух наборов, аналогичных по назначению аргументам `<набор1>` и `<набор2>` функции `sssetfirst` (в текущий момент у примитивов второго набора включены ручки и подсвечивание).

```
(ssgetfirst)
```

1.12.23. Пример работы с примитивами AutoCAD

При создании прикладного программного обеспечения часто приходится работать с примитивами рисунка, из которых нужно извлекать данные, изменять их и передавать для дальнейшего использования. Самая распространенная схема получения доступа к графическим объектам такова.

В качестве средств интерактивного выбора примитивов используются функции `entsel` или `ssget`. При необходимости сканирования рисунка для поиска в нем по определенному свойству (имени слоя, цвету и т. п.) выбор примитивов осуществляется с помощью функции `ssget`.

Функция `entsel` возвращает список, первым элементом которого является указанный пользователем примитив. Этот примитив может быть извлечен из списка с помощью функций `car` или `nth`.

Функция `ssget` возвращает имя набора, состоящего из некоторого количества имен примитивов (при неудачном выборе вместо имени набора может быть возвращено значение `nil`). Длина набора оценивается с помощью функции `sslength`, а извлечение из него нужного элемента выполняется функцией `ssname`.

Далее к полученному примитиву применяется функция `entget`, которая возвращает список с набором точечных пар, содержащих характеристики рассматриваемого примитива. Затем с помощью функций `assoc` и `cdr` из этого списка читаются нужные свойства примитива.

Проиллюстрируем приведенную схему следующим примером. Предположим, что в рисунке детали на слое HOLES находятся отверстия (круги), построенные с помощью команды CIRCLE (КРУГ). Необходимо прочитать все эти круги и сформировать два списка, в первом из которых собрать радиусы кругов, а во втором — точки с координатами центров. Примерный вариант такой программы приведен в листинге 1.18.

Листинг 1.18. Пример работы с примитивами рисунка

```
; Глава 01\Листинг 1.18.lsp
; Начальные значения списков радиусов (list_rad) и центров (list_cen)
(setq list_rad '() list_cen '())
; Создание набора из кругов на слое HOLES
(setq nab_cir (ssget "_X" (list (cons 8 "HOLES") (cons 0 "CIRCLE"))))
; Проверка, сформировался ли набор nab_cir
; (если нет, то предыдущая операция вернет nil)
(if (null nab_cir)
  (progn
    (princ "\nНет кругов на слое HOLES. "); сообщение об отсутствии
    (princ); тихий выход
  ); конец progn
  (progn
    (setq i -1 nab_len (sslength nab_cir))
; Цикл по количеству элементов набора nab_cir
    (repeat nab_len
      (setq i (1+ i))
; Выбор следующего примитива и получение его списка
      (setq cirlist (entget (ssname nab_cir i)))
      (setq radcir (cdr (assoc 40 cirlist)))
      (setq cencir (cdr (assoc 10 cirlist)))
; Добавление радиуса и точки центра к спискам list_rad и list_cen
      (setq list_rad (append list_rad (list radcir)))
      (setq list_cen (append list_cen (list cencir)))
   ); конец repeat
; Печать результирующих списков
    (princ "\nРадиусы: ")
    (princ list_rad)
    (princ "\nЦентры: ")
    (princ list_cen)
 ); конец progn
); конец if
```

Программу следует сохранить (например, в файле D:\Temp\rcl.lsp), после чего ее можно загрузить в память:

```
(load "D:\\Temp\\rcl.lsp")
```

Если файл загрузится без ошибок, то система AutoCAD его сразу выполнит.

Разберем схему работы программы. Для того чтобы выбрать из рисунка все круги на слое HOLES, используется функция `ssget` с фильтром по имени слоя и типу примитива: `(ssget "_X" (list (cons 8 "HOLES") (cons 0 "CIRCLE")))`. Результат (набор) запоминается в переменной `nab_cir`.

Количество объектов, попавших в набор `nab_cir`, вычисляется с помощью выражения `(sslength nab_cir)`. Затем с помощью функции `ssname` поочередно получаются примитивы, включенные в набор: выражение `(ssname nab_cir i)` для разных `i` возвращает примитив с номером `i-1`.

Далее формируется список с характеристиками очередного объекта-круга с помощью выражения `(entget (ssname nab_cir i))`. После этого остается только извлечь с помощью функций `cdr` и `assoc` значения радиуса (DXF-код 40) и центра (DXF-код 10). Извлекаемые значения заносятся в результирующие списки `list_rad` и `list_cen`.

Возможный вид результата работы программы:

```
Радиусы: (66.0916 29.1292)
Центры: ((179.603 194.961 0.0) (43.0525 184.539 0.0))((179.603 194.961 0.0)
(43.0525 184.539 0.0))
```

О том, как оформить эту программу в виде пользовательской функции, будет рассказано в *разд. 1.16*.

1.13. Функции доступа к табличным данным

Помимо примитивов в рисунке всегда присутствует определенный объем неграфической информации: описания блоков, таблицы слоев, таблицы видовых экранов, таблицы типов линий, таблицы размерных стилей и т. п. Все они хранятся в базе рисунка примерно в том же виде, что и примитивы.

1.13.1. Символьные таблицы

Табличные данные основных неграфических объектов рисунка сгруппированы в символьные таблицы, имеющие следующие имена:

- BLOCK — таблица описаний блоков;
- LAYER — таблица слоев;
- LTYPE — таблица типов линий;
- STYLE — таблица текстовых стилей;
- DIMSTYLE — таблица размерных стилей;
- UCS — таблица именованных систем координат;
- VIEW — таблица именованных видов;
- VPORT — таблица конфигураций видовых экранов;
- APPID — таблица имен приложений, с которыми связываются расширенные данные *(о расширенных данных см. разд. 1.14)*.

В этих таблицах находятся соответствующие табличные объекты, внутренняя организация которых аналогична организации графических примитивов, а методы доступа похожи на методы доступа к графическим примитивам. В перечне присутствуют текстовые и размерные стили. Следует обратить внимание на то, что для стилей мультилиний и стилей таблиц (как графических примитивов) применено другое средство хранения данных — словари *(о словарях см. разд. 1.15)*.

Название символьные таблицы введено разработчиком системы, оно не связано с примитивом ACAD_TABLE (таблица), а определяется именем используемого класса: `AcDbSymbolTable`.

Ниже рассматриваются функции языка LISP, осуществляющие доступ к объектам символьных таблиц.

1.13.2. Функция *tblsearch*

Получает список с характеристиками объекта таблицы по его заголовку.

```
(tblsearch <таблица> <заголовок> [<следующий>])
```

Аргументы: `<таблица>` — текстовая строка с именем таблицы, `<заголовок>` — текстовая строка с именем элемента таблицы, `<следующий>` — любое значение (важно только, `nil` или не `nil`).

Возвращаемое значение — список, соответствующий найденному объекту (или `nil`, если объект не обнаружен).

Если параметр `<следующий>` задан и не равен `nil`, то указатель функции `tblnext` устанавливается на объект таблицы с данным заголовком (следующее обращение к функции `tblnext` вернет данные об объекте, расположенном в таблице после объекта, использованного в качестве аргумента в функции `tblsearch`).

Примеры:

- `(tblsearch "LAYER" "0")` возвращает `((0 . "LAYER") (2 . "0") (70 . 0) (62 . 7) (6 . "Continuous"))`
- `(tblsearch "LTYPE" "CONTINUOUS")` возвращает `((0 . "LTYPE") (2 . "Continuous") (70 . 0) (3 . "Solid line") (72 . 65) (73 . 0) (40 . 0.0))`
- `(tblsearch "STYLE" "STANDARD")` возвращает `((0 . "STYLE") (2 . "Standard") (70 . 0) (40 . 0.0) (41 . 1.0) (50 . 0.0) (71 . 0) (42 . 2.5) (3 . "txt") (4 . ""))`

1.13.3. Функция *tblnext*

Читает характеристики очередного объекта из таблицы.

```
(tblnext <таблица> [<признак>])
```

Аргументы: `<таблица>` — текстовая строка с именем таблицы *(см. разд. 1.13.1)*, `<признак>` — любое значение (признак первого элемента таблицы — см. ниже).

Возвращаемое значение — список с характеристиками элемента таблицы или `nil`.

Если параметр `<признак>` задан и отличен от `nil`, то функция возвращает данные первого элемента таблицы (иначе возвращаются данные объекта, следующего за тем, который был прочитан в предыдущем обращении функции `tblnext` к той же таблице или в предыдущем обращении к функции `tblsearch`, вызванной с аргументом `<следующий>`, имеющим значение, отличное от `nil`). Если требуемого объекта нет (например, достигнут конец таблицы) или таблица пуста, то возвращается `nil`.

Пример:

- `(tblnext "LTYPE" T)` возвращает список с характеристиками типа линий CONTINUOUS: `((0 . "LTYPE") (2 . "Continuous") (70 . 0) (3 . "Solid line") (72 . 65) (73 . 0) (40 . 0.0))`

Пример использования функции `tblnext` совместно с функцией `tblsearch`:

- `(tblsearch "LTYPE" "CONTINUOUS" T)` возвращает `((0 . "LTYPE") (2 . "Continuous") (70 . 0) (3 . "Solid line") (72 . 65) (73 . 0) (40 . 0.0))`
- `(tblnext "LTYPE")` возвращает данные типа линий, следующего в таблице после CONTINUOUS

1.13.4. Функция *tblobjname*

Определяет имя (адрес) объекта таблицы по его заголовку (имени).

```
(tblobjname <таблица> <заголовок>)
```

Аргументы: `<таблица>` — текстовая строка с именем таблицы, `<заголовок>` — текстовая строка с именем элемента таблицы.

Возвращаемое значение — объект (его структура аналогична структуре примитива) в базе рисунка. Если объект с нужным заголовком не найден, то возвращается `nil`.

Пример:

- `(setq layer0 (tblobjname "LAYER" "0"))` возвращает `<Entity name: 7ef66c80>`

  `(entget layer0)` возвращает список `((-1 . <Entity name: 7ef66c80>) (0 . "LAYER") (330 . <Entity name: 7ef66c10>) (5 . "10") (100 . "AcDbSymbolTableRecord") (100 . "AcDbLayerTableRecord") (2 . "0") (70 . 0) (62 . 215) (420 . 10771353) (6 . "Continuous") (290 . 1) (370 . -3) (390 . <Entity name: 7ef66c78>))`

Замечание

В списке, полученном с помощью функций `tblobjname` и `entget`, больше данных по сравнению со списками, возвращаемыми функциями `tblnext` и `tblsearch`.

1.13.5. Схема работы с таблицами

Схема использования рассмотренных функций, работающих с таблицами неграфических объектов, будет ясна из следующих примеров.

- `(setq lay1 (tblnext "LAYER" T))` возвращает список с данными первого слоя рисунка (первым слоем должен быть слой с именем `"0"`): `((0 . "LAYER") (2 . "0") (70 . 0) (62 . 7) (6 . "Continuous"))`

В этом списке в точечной паре с DXF-кодом 0 находится имя таблицы, в которой ищется неграфический объект, с кодом 2 — имя объекта (в нашем случае слоя), с кодом 6 — тип линий примитивов слоя по умолчанию (т. е. каким типом отображать объекты с типом линий ПоСлою (ByLayer)), с кодом 62 — цвет слоя по умолчанию, 70 — состояние слоя по отношению к операциям блокирования, замораживания. Нетрудно заметить, что аналогичные DXF-коды используются в списке, возвращаемом функцией `entget` для примитивов.

- `(setq lay2 (tblnext "LAYER"))` возвращает список с данными второго слоя рисунка, например: `((0 . "LAYER") (2 . "Стена1") (70 . 4) (62 . 253) (6 . "BORDER"))`
- `(setq lay2ent (tblobjname "LAYER" "Стена1"))` возвращает `<Entity name: 7ef66f98>`

Теперь для объекта `lay2ent` с помощью функции `entget` можно получить список с данными.

Применяя функции `tblnext`, `tblobjname` и `tblsearch`, можно проанализировать наличие в рисунке таких элементов, как описания блоков, пользовательские системы координат или текстовые стили с определенными именами, и выбрать соответствующий вариант работы программы.

1.13.6. Функция *setview*

Следующая функция работает с символьной таблицей VIEW, устанавливая вид в нужном видовом экране.

```
(setview <вид> [<экран>])
```

Аргументы: `<вид>` — список с данными именованного вида в той форме, в какой он возвращается функциями `tblnext` и `tblsearch` для таблицы VIEW; `<экран>` — номер видового экрана в соответствии со значением системной переменной CVPORTS (0 соответствует текущему видовому экрану).

Возвращаемое значение — значение аргумента `<вид>` при успешном выполнении или `nil` — при ошибке.

Рассмотрим работу функции `setview` в пространстве модели на следующем примере, когда в рисунке создано три именованных вида: ISO, TOP и FRONT (см. листинг 1.19).

Листинг 1.19. Пример работы с видами

```
; Глава 01\Листинг 1.19.lsp
; Получение данных вида ISO
(setq lviso (tblsearch "VIEW" "ISO"))
; возвращает ((0 . "VIEW") (2 . "ISO") (70 . 0) (40 . 642.839)
; (10 43.4871 146.357) (41 . 1141.07) (11 -1.0 -1.0 1.0)
; (12 0.0 0.0 0.0) (42 . 50.0) (43 . 0.0) (44 . 0.0) (50 . 0.0)
; (71 . 0) (72 . 1) (110 0.0 0.0 0.0) (111 1.0 0.0 0.0)
; (112 0.0 0.0 1.0) (79 . 3) (146 . 0.0))
; Получение данных вида TOP
(setq lvtop (tblsearch "VIEW" "TOP"))
; возвращает ((0 . "VIEW") (2 . "TOP") (70 . 0) (40 . 300.362)
; (10 210.0 148.5) (41 . 533.157) (11 0.0 0.0 1.0)
; (12 0.0 0.0 0.0) (42 . 50.0) (43 . 0.0) (44 . 0.0) (50 . 0.0)
; (71 . 0) (72 . 1) (110 0.0 0.0 0.0) (111 1.0 0.0 0.0)
; (112 0.0 1.0 0.0) (79 . 1) (146 . 0.0))
; Получение данных вида FRONT
(setq lvfro (tblsearch "VIEW" "FRONT"))
; возвращает ((0 . "VIEW") (2 . "FRONT") (70 . 0) (40 . 424.755)
; (10 210.0 0.0) (41 . 753.96) (11 0.0 -1.0 0.0)
; (12 0.0 0.0 0.0) (42 . 50.0) (43 . 0.0) (44 . 0.0) (50 . 0.0)
; (71 . 0) (72 . 1) (110 0.0 0.0 0.0) (111 1.0 0.0 0.0)
; (112 0.0 0.0 1.0) (79 . 3) (146 . 0.0))
; Установка в видовом экране 2 пространства модели
; именованного вида ISO
(setview lviso 2)
; Установка в видовом экране 5 пространства модели
; именованного вида TOP
(setview lvtop 5)
; Установка в видовом экране 0 пространства модели
; именованного вида FRONT
(setview lvfro 0)
```

Сначала с помощью функции `tblsearch` извлекаются данные видов с именами ISO, TOP и FRONT и сохраняются в переменных `lviso`, `lvtop` и `lvfro`. При этом в качестве значений возвращаются списки примерно следующего содержания: `((0 . "VIEW") (2 . "ISO") (70 . 0) (40 . 642.839) (10 43.4871 146.357) (41 . 1141.07) (11 -1.0 -1.0 1.0) (12 0.0 0.0 0.0) (42 . 50.0) (43 . 0.0) (44 . 0.0) (50 . 0.0) (71 . 0) (72 . 1) (110 0.0 0.0 0.0) (111 1.0 0.0 0.0) (112 0.0 0.0 1.0) (79 . 3) (146 . 0.0))`.

Если в пространстве модели разделить графический экран на несколько видовых экранов, то каждому из них будет соответствовать свое значение системной переменной CVPORT. При активизации очередного видового экрана системная переменная CVPORT будет менять свое значение (попробуйте поменять активные экраны и прочитать значение системной переменной).

Пусть деление графического экрана выполнено на четыре части, тогда созданным видовым экранам будут соответствовать, например, такие значения CVPORT: 2, 3, 4, 5. Теперь в каждом из видовых экранов по номеру можно задать именованный вид.

В последних строках листинга 1.19 трижды применяется функция `setview` для установки видов: `(setview lviso 2)` задает вид ISO в видовом экране с номером 2, `(setview lvtop 5)` — вид TOP в экране 5 и `(setview lvfro 0)` — вид FRONT в текущем видовом экране (текущий экран имеет номер 0 и им может оказаться любой из четырех экранов).

Аналогично может быть выполнена установка видов в видовых экранах пространства листа.

1.13.7. Функция *snvalid*

Проверяет имя на возможность использования его в качестве элемента символьной таблицы (чтобы исключить, например, нелегальные имена слоев, типов линий и т. п.).

```
(snvalid <имя> [<флаг>])
```

Аргументы: `<имя>` — текстовая строка, `<флаг>` — целое число (0 или 1).

Возвращаемое значение — `T`, если имя является допустимым, и `nil` — в противном случае.

Аргумент `<флаг>` управляет возможностью использования символа `"|"`. Если аргументу `<флаг>` задано значение 0, то символ считается недопустимым. Если задано значение 1, то символ может встретиться в строке, но он не должен быть первым или последним (применяется в именах слоев внешних ссылок).

К недопустимым всегда относятся пустые строки. Кроме того, в именах не могут участвовать управляющие и графические символы.

Функция `snvalid` работает в двух режимах, в зависимости от текущего значения системной переменной EXTNAMES. Если системная переменная EXTNAMES имеет значение 0, то тогда имена проверяются с теми же ограничениями, что и в версиях системы AutoCAD, предшествовавших AutoCAD 2000: допускаются буквы, цифры и символы доллара (`$`), подчеркивания (`_`) и дефиса (`-`).

Если переменная EXTNAMES имеет значение 1 (а оно действует по умолчанию), то функция `snvalid` при проверке имени предъявляет менее жесткие требования. По-прежнему, к недопустимым относятся управляющие и графические символы, а также следующие: `<`, `>`, `/`, `\`, `"` (двойная кавычка), `:`, `?`, `*`, , (запятая), `=`, `` ` ``, ; (символ с кодом 59). Символ пробела является допустимым.

Примеры (для случая, когда EXTNAMES имеет значение 1):

- `(snvalid "chamfer@")` возвращает `T`
- `(snvalid "chamfer*")` возвращает `nil`

- `(snvalid "cham|fer")` возвращает `nil`
- `(snvalid "cham fer")` возвращает `T`
- `(snvalid "cham|fer" 1)` возвращает `T`

Примеры (для случая, когда EXTNAMES имеет значение 0):

- `(snvalid "chamfer@")` возвращает `nil`
- `(snvalid "chamfer*")` возвращает `nil`
- `(snvalid "cham|fer")` возвращает `nil`
- `(snvalid "cham fer")` возвращает `nil`
- `(snvalid "cham|fer" 1)` возвращает `T`

1.14. Функции, работающие с расширенными данными

К примитивам и неграфическим объектам рисунка можно добавлять *расширенные данные*, т. е. текстовые или числовые данные, которые желательно хранить вместе с объектом (например, марки материалов, наименования узлов, секций и т. п.). Обычно наличие такой информации позволяет разрабатывать программы связи с базами данных (рассмотрение таких программ не является предметом настоящей книги). Расширенные данные связываются с именем определенного приложения (приложение может быть чисто условным). У одного графического примитива или неграфического объекта могут быть данные нескольких приложений, которые располагаются последовательно в определенной структуре.

Объем расширенных данных одного объекта ограничен.

1.14.1. Структура расширенных данных

Если расширенные данные присоединены к примитиву (или неграфическому объекту), то в базе рисунка они следуют за обычными данными (DXF-коды этих данных находятся в диапазоне от –2 до 239). Извлечь расширенные данные можно с помощью функции `entget`, при обращении к ней нужно указывать имя приложения, с которым они связаны (см. описание функции `entget` в *разд. 1.12.5*). В списке, возвращаемом функцией `entget`, расширенные данные находятся в подсписке, начинающемся с DXF-кода –3 (признак наличия расширенных данных), в котором далее располагаются точечная пара с кодом 1001 (начало расширенных данных и имя приложения, с которым они связываются) и точечные пары с DXF-кодами 1000, 1002—1071. Назначение DXF-кодов расширенных данных описывается в табл. 1.8.

Как было сказано выше, расширенные данные связываются с именем конкретного приложения. Имена приложений хранятся в рисунке в символьной таблице

APPID *(см. разд. 1.13.1)*. Чтобы занести имя приложения в эту таблицу, следует пользоваться функцией `regapp` *(см. разд. 1.14.2)*.

Таблица 1.8. *Назначение DXF-кодов расширенных данных*

Код	Назначение
1000	Строковая константа длиной не более 255 знаков (256-й знак зарезервирован под символ с кодом 0)
1001	Строковая константа, используемая только для указания имени приложения, длиной не более 31 знака (если код 1001 попадает внутрь списка, организуемого с помощью DXF-кодов 1002, то он интерпретируется как обычная константа, а не имя приложения; см. далее описание DXF-кода 1002). Имя приложения должно состоять только из цифр, латинских букв и символов `$`, `-`, `_`
1002	Строка, имеющая значения "`{`" или "`}`", которые носят тот же смысл, что и левая и правая круглые скобки в обозначении списка. С помощью группы с кодом 1002 можно формировать в расширенных данных списковые структуры (они должны начинаться открывающей скобкой с кодом 1002, затем должны идти данные, включаемые в список, а закрывающая скобка с кодом 1002 завершает оформление списка). См. также замечание об интерпретации данных группы 1001 в таком списке
1003	Имя слоя, с которым связываются расширенные данные
1004	Двоичные данные длиной не более 127 байтов. Обрабатываются только в ObjectARX
1005	Метка объекта в базе чертежа
1010	Трехмерная точка (три вещественных числа)
1011	Трехмерная точка, интерпретируемая как положение в МСК. Такая точка перемещается, масштабируется, поворачивается и симметрируется одновременно с родительским примитивом рисунка. Если точка попадает в рамку выбора команды STRETCH (РАСТЯНУТЬ), то она подвергается операции растягивания вместе с родительским примитивом
1012	Трехмерная точка, интерпретируемая как перемещение в МСК. Такая точка не перемещается и не растягивается одновременно с основным примитивом, но участвует вместе с ним в операциях масштабирования, поворота и симметрирования
1013	Трехмерная точка, интерпретируемая как направление в МСК. Такая точка не масштабируется, не перемещается и не растягивается одновременно с основным примитивом, но участвует вместе с ним в операциях поворота и симметрирования. Вектор с координатами, взятыми из группы с кодом 1013, является нормализованным, т. е. единичной длины
1040	Вещественное число
1041	Вещественное число, интерпретируемое как расстояние. Оно масштабируется одновременно с родительским примитивом
1042	Вещественное число, интерпретируемое как масштабный коэффициент. Масштабируется одновременно с родительским примитивом

Таблица 1.8 *(окончание)*

Код	Назначение
1070	16-битовое целое число (со знаком или без знака)
1071	32-битовое целое число со знаком (длинное целое). Хотя система AutoCAD хранит данные группы с кодом 1071 как длинное целое, LISP возвращает их как вещественные числа. Функциям `entmake` и `entmod` в точечной паре с кодом 1071 можно передавать либо целые числа, либо вещественные. ObjectARX трактует данные группы 1071 как длинные целые

1.14.2. Функция *regapp*

Заносит имя приложения в таблицу APPID.

```
(regapp <приложение>)
```

Параметр `<приложение>` должен быть текстовой строкой (по возможности нужно выбирать имя приложения таким, чтобы оно не могло совпадать с именами приложений других разработчиков — иначе это приведет к неправильной работе с расширенными данными).

Функция `regapp` возвращает имя приложения (или `nil` в случае ошибки). Значение `nil` возвращается также, если приложение с таким именем уже регистрировалось.

Пример:

- `(regapp "SHIPDESIGN")` возвращает `"SHIPDESIGN"`

1.14.3. Присоединение расширенных данных

Функция `entmake` *(см. разд. 1.12.6)* может создать примитив и сразу присоединить к нему расширенные данные. С помощью функции `entmod` можно добавить расширенные данные к существующему графическому объекту.

Предположим в качестве примера, что к последнему существующему примитиву рисунка (пусть им является отрезок) нужно добавить расширенные данные, связанные с приложением "LAUNCH01". Соответствующие действия приведены в листинге 1.20.

Листинг 1.20. Пример присоединения расширенных данных

```
; Глава 01\Листинг 1.20.lsp
; Регистрация приложения LAUNCH01
(regapp "LAUNCH01")
; Получение списка данных последнего примитива
(setq le (entget (entlast)))
```

```
; Формирование подсписка расширенных данных
(setq edata (list (list -3 (list "LAUNCH01"
  (cons 1000 "Ступень") (cons 1070 121)))))
; Включение в список расширенных данных
(setq le2 (append le edata))
; Присоединение расширенных данных к примитиву
; (обновление примитива)
(entmod le2)
; Проверка данных примитива
(entget (entlast) '("LAUNCH01"))
```

Вначале необходимо зарегистрировать имя приложения LAUNCH01, с которым связываются присоединяемые расширенные данные. Повторная регистрация системой не выполняется, но и ошибки не вызывает.

В переменной `le` сохраняется список данных последнего примитива рисунка (перед выполнением программы у него нет расширенных данных). Затем создается список `edata`, который имеет структуру расширенных данных:

```
(setq edata (list (list -3 (list "LAUNCH01" (cons 1000 "Ступень")
 (cons 1070 121)))))
```

Здесь DXF-код –3 задает начало расширенных данных, в которые входят строка с именем приложения и две точечные пары: с кодом 1000 (строка) и с кодом 1070 (целое число).

Далее с помощью функции `append` формируется список `le2`, который является объединением списков `le` и `edata`. Примерный вид этого списка таков:

```
((-1 . <Entity name: 7ef7f240>) (0 . "LINE") (330 . <Entity name: 7ef66cf8>)
(5 . "138") (100 . "AcDbEntity") (67 . 0) (410 . "Model")
(8 . "0") (100 . "AcDbLine") (10 -402.145 0.0 591.48)
(11 291.58 0.0 81.7246) (210 0.0 -1.0 0.0) (-3 ("LAUNCH01"
(1000 . "Ступень") (1070 . 121))))
```

Непосредственное присоединение расширенных данных к примитиву осуществляется путем обновления этого примитива с помощью функции `entmod` и списка `le2`.

Размер расширенных данных любого примитива не должен превышать 16 Кбайт –1 (16 383 байта). Поэтому по мере добавления таких данных их размер нужно контролировать. Этой цели служат рассматриваемые ниже функции `xdsize` и `xdroom`.

Перед записью в зону примитива новой порции расширенных данных следует проверить, достаточно ли для них места.

Об удалении расширенных данных *см. приложение 7*.

1.14.4. Функция *xdsize*

Получает размер в байтах, необходимый для сохранения расширенных данных.

```
(xdsize <список>)
```

Аргумент `<список>` — это список с расширенными данными, начинающийся с DXF-кода –3.

Возвращаемое значение — целое число (размер в байтах).

Пример:

- `(xdsize '(-3 ("LAUNCH01" (1000 . "Ступень") (1070 . 121))))` возвращает `15`

1.14.5. Функция *xdroom*

Вычисляет количество свободных байтов в зоне расширенных данных примитива.

```
(xdroom <примитив>)
```

Аргумент — имя примитива.

Возвращаемое значение — целое число. При ошибке возвращается `nil`.

Пример:

- `(xdroom (entlast))` возвращает `16383`, если у последнего примитива еще нет расширенных данных

1.14.5.1. О примитиве VIEWPORT

Примитив VIEWPORT (видовой экран пространства листа) является исключением из общего правила. Если выполнить выражение `(xdroom evp)` для видового экрана `evp`, то получим результат 16383. Казалось бы, примитив не имеет расширенных данных, т. к. формально доступны все 16383 байта для записи расширенных данных пользователя.

Однако попробуем выполнить следующее выражение и получить все расширенные данные видового экрана (для этого в качестве имени приложения используем звездочку):

```
(entget evp '("*"))
```

Получим неожиданный результат: `((-1 . <Entity name: 7ef66f90>) (0 . "VIEWPORT") (330 . <Entity name: 7ef66ed0>) (5 . "EA") (100 . "AcDbEntity") (67 . 1) (410 . "Layout1") (8 . "0") (100 . "AcDbViewport") (10 61.6814 148.552 0.0) (40 . 71.9628) (41 . 53.8956) (68 . 2) (69 . 2) (12 71.8247 254.072 0.0) (13 0.0 0.0 0.0) (14 10.0 10.0 0.0) (15 10.0 10.0 0.0) (16 0.0 0.0 1.0) (17 0.0 0.0 0.0) (42 . 50.0) (43 . 0.0) (44 . 0.0) (45 . 111.452) (50 . 0.0) (51 . 0.0) (72 . 1000) (90 . 32864) (281 . 0) (71 . 1) (74 . 0) (110 0.0 0.0 0.0) (111 1.0 0.0 0.0) (112 0.0 1.0 0.0) (79 . 0) (146 . 0.0) (170 .`

```
0) (-3 ("ACAD" (1000 . "MVIEW") (1002 . "{") (1070 . 16) (1010 0.0 0.0 0.0)
(1010 0.0 0.0 1.0) (1040 . 0.0) (1040 . 111.452) (1040 . 71.8247) (1040 .
254.072) (1040 . 50.0) (1040 . 0.0) (1040 . 0.0) (1070 . 0) (1070 . 1000)
(1070 . 1) (1070 . 3) (1070 . 0) (1070 . 0) (1070 . 0) (1070 . 0) (1040 . 0.0)
(1040 . 0.0) (1040 . 0.0) (1040 . 10.0) (1040 . 10.0) (1040 . 10.0) (1040 .
10.0) (1070 . 0) (1002 . "{") (1002 . "}") (1002 . "}")))
```

Оказывается, видовой экран имеет расширенные данные, связанные с приложением ACAD (т. е. данные самой системы AutoCAD), и функция `xdroom` на них не реагирует! В предыдущем абзаце эти данные выделены полужирным шрифтом.

Казалось бы, можно попытаться добавить к примитиву свои расширенные данные. Однако вспомним: функция `entmod` не работатет с примитивом VIEWPORT.

Вывод: с видовым экраном **нельзя** связать пользовательские расширенные данные.

1.15. Функции, работающие со словарями и X-записями

Свойствами, аналогичными таблицам, обладают в рисунке объекты, называемые *словарями* (dictionaries). Пользовательские приложения могут создавать свои словари. Словарь можно привязать в первую очередь к документу (рисунку). Однако словарь может быть и у графического примитива, семейства, слоя и т. д.

Функции, рассматриваемые в следующих разделах, выполняют операции над словарями и их элементами.

1.15.1. Словари

С помощью словарей система AutoCAD расширяет круг хранения своих данных. К наиболее понятным пользователю словарям, привязанным к рисунку, можно отнести, например, следующие:

- стилей мультилиний;
- групп;
- растровых изображений;
- масок (примитивов WIPEOUT);
- фильтров слоев;
- листов чертежа;
- стилей печати;
- стилей таблиц.

У словарей, связанных с рисунком, есть корневой словарь (своего рода оглавление), который содержит другие словари рисунка. Имя (адрес) корневого словаря можно получить с помощью функции `namedobjdict`.

Для работы со словарями рисунка используются функции `dictsearch`, `dictnext`, `dictadd`, `dictrename`, `dictremove`. Приведена также функция `layoutlist`, облегчающая работу со словарем вкладок листов чертежа.

Для работы со словарями примитивов и других объектов следует использовать рассматриваемые в *главах 6—8* средства ActiveX (функции `vlax-ldata-list`, `vlax-ldata-put`, `vlax-ldata-get`, `vlax-ldata-delete`, `vlax-ldata-test`, свойство `HasExtensionDictionary`, метод `GetExtensionDictionary`).

1.15.2. Функция *namedobjdict*

Получение имени корневого словаря рисунка.

```
(namedobjdict)
```

Возвращаемое значение — словарь как объект базы данных рисунка, аналогичный примитиву или символьной таблице (или более точно, указатель на словарь).

Пример:

- `(namedobjdict)` возвращает `<Entity name: 7ef88c60>`

Возвращенный объект является словарем, содержащим имена словарей рисунка.

В листинге 1.21 приводится пример, с помощью которого можно извлечь эти имена.

Листинг 1.21. Пример получения имен словарей рисунка

```
; Глава 01\Листинг 1.21.lsp
; Корневой словарь
(setq od (namedobjdict))
; Чтение данных объекта od
(setq lod (entget od))
; Выделение имен словарей
(mapcar 'cdr (vl-remove-if-not
  (function (lambda (x) (= (car x) 3))) lod))
```

В рассматриваемом примере функция `entget`, примененная к корневому словарю `od`, для нового рисунка, созданного в системе AutoCAD 2006, возвращает примерно следующий список:

```
((-1 . <Entity name: 7ef88c60>) (0 . "DICTIONARY")
(330 . <Entity name: 0>) (5 . "C") (100 . "AcDbDictionary") (280 . 0)
(281 . 1) (3 . "ACAD_COLOR") (350 . <Entity name: 7ef88dd8>)
(3 . "ACAD_GROUP") (350 . <Entity name: 7ef88c68>) (3 . "ACAD_LAYOUT")
(350 . <Entity name: 7ef88cd0>) (3 . "ACAD_MATERIAL")
(350 . <Entity name: 7ef88dd0>) (3 . "ACAD_MLINESTYLE")
(350 . <Entity name: 7ef88cb8>) (3 . "ACAD_PLOTSETTINGS")
(350 . <Entity name: 7ef88cc8>) (3 . "ACAD_PLOTSTYLENAME")
```

```
(350 . <Entity name: 7ef88c70>) (3 . "ACAD_TABLESTYLE")
(350 . <Entity name: 7ef88e30>) (3 . "AcDbVariableDictionary")
(350 . <Entity name: 7ef88d70>))
```

В этом списке точечные пары с DXF-кодом 3 содержат имена словарей рисунка. Последнее выражение листинга извлекает из этого списка значения этих точечных пар (по поводу примененных функций `function` и `lambda` *см. разд. 1.16*) и возвращает:

```
("ACAD_COLOR" "ACAD_GROUP" "ACAD_LAYOUT" "ACAD_MATERIAL" "ACAD_MLINESTYLE"
"ACAD_PLOTSETTINGS" "ACAD_PLOTSTYLENAME" "ACAD_TABLESTYLE"
"AcDbVariableDictionary")
```

Таким образом мы получили для нового пустого рисунка имена созданных в нем словарей. В процессе редактирования документа система может создать в нем и другие словари. Расшифруем названия тех словарей, которые мы получили:

- ACAD_COLOR — словарь цветов как объектов;
- ACAD_GROUP — словарь групп;
- ACAD_LAYOUT — словарь вкладок листов;
- ACAD_MATERIAL — словарь материалов;
- ACAD_MLINESTYLE — словарь стилей мультилиний;
- ACAD_PLOTSETTINGS — словарь настроек печати;
- ACAD_PLOTSTYLENAME — словарь стилей печати;
- ACAD_TABLESTYLE — словарь стилей таблиц;
- AcDbVariableDictionary — словарь хранения некоторых системных переменных.

1.15.3. Функция *dictsearch*

Читает данные элемента словаря (аналогично функции `tblsearch`, работающей с символьными таблицами).

```
(dictsearch <словарь> <заголовок> [<следующий>])
```

Аргументы: `<словарь>` — словарь или объект базы рисунка, `<заголовок>` — строка с именем элемента словаря, `<следующий>` — признак чтения следующего за указанным в предыдущем аргументе элемента в ближайшем вызове функции `dictnext`.

Возвращаемое значение — список с данными элемента словаря.

Воспользуемся примером из *разд. 1.15.2* и сформированной в нем переменной `od`, содержащей корневой словарь рисунка. С помощью функции `dictsearch` можно получить списки с данными некоторых словарей.

Пример (получение данных словаря вкладок листов):

♦ `(dictsearch od "ACAD_LAYOUT")` возвращает
```
((-1 . <Entity name: 7ef66cd0>) (0 . "DICTIONARY") (5 . "1A")
(102 . "{ACAD_REACTORS") (330 . <Entity name: 7ef66c60>) (102 . "}")
(330 . <Entity name: 7ef66c60>) (100 . "AcDbDictionary") (280 . 0)
(281 . 1) (3 . "Layout1") (350 . <Entity name: 7ef66ed8>)
(3 . "Layout2") (350 . <Entity name: 7ef66ef8>)
(3 . "Model") (350 . <Entity name: 7ef66d10>))
```

Из точечных пар с DXF-кодом 3 этого списка можно извлечь имена вкладок листов: `Layout1`, `Layout2` (сравните с рассматриваемой в *разд. 1.15.8* функцией `layoutlist`). В списке вместе с именами листов фигурирует и имя вкладки пространства модели (`Model`).

Пример (получение данных словаря стилей мультилиний):

♦ `(dictsearch od "ACAD_MLINESTYLE")` возвращает
```
((-1 . <Entity name: 7ef66cb8>) (0 . "DICTIONARY") (5 . "17")
(102 . "{ACAD_REACTORS") (330 . <Entity name: 7ef66c60>) (102 . "}")
(330 . <Entity name: 7ef66c60>) (100 . "AcDbDictionary") (280 . 0)
(281 . 1) (3 . "Standard") (350 . <Entity name: 7ef66cc0>))
```

Возвращенное значение показывает, что в новом рисунке имеется только один стиль мультилиний — `Standard`.

Пример (получение данных словаря стилей печати):

♦ `(dictsearch od "ACAD_PLOTSTYLENAME")` возвращает
```
((-1 . <Entity name: 7ef66c70>) (0 . "ACDBDICTIONARYWDFLT") (5 . "E")
(102 . "{ACAD_REACTORS") (330 . <Entity name: 7ef66c60>) (102 . "}")
(330 . <Entity name: 7ef66c60>) (100 . "AcDbDictionary") (280 . 0)
(281 . 1) (3 . "Normal") (350 . <Entity name: 7ef66c78>)
(100 . "AcDbDictionaryWithDefault") (340 . <Entity name: 7ef66c78>))
```

Приведенные примеры показывают, что словари как объекты могут быть разных типов. Об этом говорят отличающиеся значения, записанные в точечные пары с DXF-кодом 0 (нам встретились `DICTIONARY` и `ACDBDICTIONARYWDFLT`).

1.15.4. Функция *dictnext*

Читает очередной элемент словаря (аналогична функции `tblnext`, но работает со словарями).

```
(dictnext <словарь> [<признак>])
```

Аргументы: `<словарь>` — словарь (объект базы рисунка), `<признак>` — любое значение (важно только, равно оно `nil` или нет).

Возвращаемое значение — список с данными элемента словаря. Если был достигнут конец словаря, то возвращается `nil`.

Если аргумент `<признак>` задан и не равен `nil`, то функция читает первый элемент словаря. В других случаях функция `dictnext` читает следующий элемент (преды-

дущим был элемент, прочитанный либо в предыдущем вызове функции `dictnext`, либо функцией `dictsearch`, вызванной с аргументом `<следующий>`, имеющим значение, отличное от `nil`).

Если ваша программа работает сразу с несколькими словарями, то следует иметь в виду, что переход к другому словарю теряет положение указателя чтения предыдущего словаря.

В листинге 1.22 приведен пример чтения данных вкладок листов.

Листинг 1.22. Пример чтения данных вкладок листов

```
; Глава 01\Листинг 1.22.lsp
; Получение данных словаря ACAD_LAYOUT
(setq dbegin (dictsearch (namedobjdict) "ACAD_LAYOUT"))
; Объект словаря вкладок
(setq layoutdict (cdr (assoc -1 dbegin)))
; Получение данных первой вкладки
(setq lt1 (dictnext layoutdict T))
; Получение данных второй вкладки
(setq lt2 (dictnext layoutdict))
```

В предыдущем разделе мы уже видели, из чего состоит список `dbegin` с данными словаря ACAD_LAYOUT. С помощью DXF-кода –1 в переменную `layoutdict` извлекаем указатель на сам словарь.

Затем выражение `(dictnext layoutdict T)` возвращает список с данными первой вкладки рисунка (ей в нашем примере является вкладка `Layout1`):

```
((-1 . <Entity name: 7ef66ed8>) (0 . "LAYOUT") (5 . "D3") (102 .
"{ACAD_REACTORS") (330 . <Entity name: 7ef66cd0>) (102 . "}") (330 . <Entity
name: 7ef66cd0>) (100 . "AcDbPlotSettings") (1 . "") (2 . "C:\\Documents and
Settings\\basas\\Application Data\\Autodesk\\AutoCAD
2005\\R16.2\\enu\\plotters\\Default Windows System Printer.pc3") (4 . "") (40
. 0.0) (41 . 0.0) (42 . 0.0) (43 . 0.0) (44 . 0.0) (45 . 0.0) (46 . 0.0) (47 .
0.0) (48 . 0.0) (49 . 0.0) (140 . 0.0) (141 . 0.0) (142 . 1.0) (143 . 1.0) (70
. 688) (72 . 0) (73 . 0) (74 . 5) (7 . "") (75 . 16) (147 . 1.0) (76 . 0) (77
. 2) (78 . 300) (148 . 0.0) (149 . 0.0) (100 . "AcDbLayout") (1 . "Layout1")
(70 . 1) (71 . 1) (10 0.0 0.0 0.0) (11 12.0 9.0 0.0) (12 0.0 0.0 0.0) (14 0.0
0.0 0.0) (15 0.0 0.0 0.0) (146 . 0.0) (13 0.0 0.0 0.0) (16 1.0 0.0 0.0) (17
0.0 1.0 0.0) (76 . 0) (330 . <Entity name: 7ef66ed0>))
```

Выражение `(dictnext layoutdict)` возвращает список с данными второй вкладки (`Layout2`):

```
((-1 . <Entity name: 7ef66ef8>) (0 . "LAYOUT") (5 . "D7")
(102 . "{ACAD_REACTORS") (330 . <Entity name: 7ef66cd0>) (102 . "}")
(330 . <Entity name: 7ef66cd0>) (100 . "AcDbPlotSettings") (1 . "")
(2 . "C:\\Documents and Settings\\basas\\Application Data\\Autodesk
\\AutoCAD 2006\\R16.2\\enu\\plotters\\Default Windows System Printer.pc3")
(4 . "") (40 . 0.0) (41 . 0.0) (42 . 0.0) (43 . 0.0) (44 . 0.0) (45 . 0.0)
```

```
(46 . 0.0) (47 . 0.0) (48 . 0.0) (49 . 0.0) (140 . 0.0) (141 . 0.0) (142 .
1.0) (143 . 1.0) (70 . 688) (72 . 0) (73 . 0) (74 . 5) (7 . "") (75 . 16) (147
. 1.0) (76 . 0) (77 . 2) (78 . 300) (148 . 0.0) (149 . 0.0) (100 .
"AcDbLayout") (1 . "Layout2") (70 . 1) (71 . 2) (10 0.0 0.0 0.0) (11 12.0 9.0
0.0) (12 0.0 0.0 0.0) (14 0.0 0.0 0.0) (15 0.0 0.0 0.0) (146 . 0.0) (13 0.0
0.0 0.0) (16 1.0 0.0 0.0) (17 0.0 1.0 0.0) (76 . 0) (330 . <Entity name:
7ef66ef0>))
```

Из рассмотренного примера видно, что имена вкладок хранятся в точечных парах с DXF-кодом 1 (см. также функцию `layoutlist` в *разд. 1.15.8*).

1.15.5. Функция *dictadd*

Добавляет новый объект к словарю.

```
(dictadd <словарь> <имя> <объект>)
```

Аргументы: `<словарь>` — словарь как объект (примитив), `<имя>` — строка (ключевое имя элемента, добавляемого к словарю), `<объект>` — объект, в котором находится значение элемента (нового объекта как элемента базы рисунка).

Возвращаемое значение — примитив добавленного объекта.

Значения, записываемые в словарь, хранятся в виде пары: имя (ключ) и непосредственно данные. Поиск в словаре производится по ключевому имени, которое должно быть строкой и не должно повторяться внутри одного словаря.

Для создания новых элементов словарей используется функция `entmakex`. При создании новых элементов словаря следует предварительно убедиться, что элемента с таким же именем нет в словаре. Повторение имен может вызывать неправильную работу программы (возможно даже зацикливание).

Для создания пользовательского словаря следует с помощью функции `dictadd` добавить объект словаря к корневому словарю. Функция `dictadd` служит и для добавления данных к пользовательским словарям. Такие данные оформляются в виде X-записей, которые рассматриваются в *разд. 1.15.9*.

В листинге 1.23 приведен пример добавления нового словаря к корневому словарю.

Листинг 1.23. Пример создания пользовательского словаря

```
; Глава 01\Листинг 1.23.lsp
; Создание списка данных будущего словаря
(setq diclist '((0 . "DICTIONARY") (100 . "AcDbDictionary")))
; Создание объекта DICTIONARY для будущего словаря
(setq newdic (entmakex diclist))
; Добавление словаря с именем к корневому словарю
(dictadd (namedobjdict) "МОЙ_СЛОВАРЬ" newdic)
```

Пользовательские словари являются объектами типа DICTIONARY и в базе рисунка относятся к классу `AcDbDictionary`. Поэтому список с данными должен содержать точечные пары с соответствующими DXF-кодами (0 и 100): `(setq diclist '((0 . "DICTIONARY") (100 . "AcDbDictionary")))`.

По этому списку с помощью функции создается новый объект базы рисунка (пока этот объект ни с чем не связан): выражение `(setq newdic (entmakex diclist))` возвращает `<Entity name: 7ef66f68>`.

Теперь заготовку словаря необходимо добавить в корневой словарь с именем МОЙ_СЛОВАРЬ. Выражение `(dictadd (namedobjdict) "МОЙ_СЛОВАРЬ" newdic)` добавляет элемент к корневому словарю, создавая, таким образом, новый пользовательский словарь, привязанный к текущему рисунку.

Чтобы прочитать данные нового объекта, воспользуемся функцией `entget`: `(entget newdic)` возвращает `((-1 . <Entity name: 7ef66f68>) (0 . "DICTIONARY") (5 . "E5") (102 . "{ACAD_REACTORS") (330 . <Entity name: 7ef66c60>) (102 . "}") (330 . <Entity name: 7ef66c60>) (100 . "AcDbDictionary") (280 . 0) (281 . 1))`. Понятно, это тот же список `diclist`, но дополненный служебными точечными парами.

Наш словарь должен одновременно попасть с именем МОЙ_СЛОВАРЬ в корневой словарь. Проверим это опять же с помощью функции `entget`:

```
(entget (namedobjdict)) возвращает ((-1 . <Entity name: 7ef66c60>)
(0 . "DICTIONARY") (330 . <Entity name: 0>) (5 . "C") (100 . "AcDbDictionary")
(280 . 0) (281 . 1) (3 . "ACAD_COLOR") (350 . <Entity name: 7ef66dd8>)
(3 . "ACAD_GROUP") (350 . <Entity name: 7ef66c68>) (3 . "ACAD_LAYOUT")
(350 . <Entity name: 7ef66cd0>) (3 . "ACAD_MATERIAL") (350 . <Entity name:
7ef66dd0>) (3 . "ACAD_MLINESTYLE") (350 . <Entity name: 7ef66cb8>)
(3 . "ACAD_PLOTSETTINGS") (350 . <Entity name: 7ef66cc8>)
(3 . "ACAD_PLOTSTYLENAME") (350 . <Entity name: 7ef66c70>)
(3 . "ACAD_TABLESTYLE") (350 . <Entity name: 7ef66e30>)
(3 . "AcDbVariableDictionary") (350 . <Entity name: 7ef66d70>)
(3 . "МОЙ_СЛОВАРЬ") (350 . <Entity name: 7ef66f68>))
```

Как видим из полученного списка, в данных корневого словаря появилась точечная пара `(3 . "МОЙ_СЛОВАРЬ")` с именем нашего словаря — словарь добавлен. При создании словарей пользователя следует иметь в виду, что префикс `ACAD_` зарезервирован для словарей, создаваемых системой AutoCAD.

1.15.6. Функция *dictremove*

Удаляет элемент словаря.

```
(dictremove <словарь> <имя>)
```

Аргументы: `<словарь>` — словарь как объект, `<имя>` — ключевое имя элемента, удаляемого из словаря.

Возвращаемое значение — удаленный объект как элемент базы рисунка.

Пример (удаление словаря из корневого словаря рисунка — продолжение примера из *разд. 1.15.5*):

- `(dictremove (namedobjdict) newdic)` возвращает `<Entity name: 7ef66f68>`

Функция `dictremove` удаляет только ссылки на объекты, использованные в словаре. Поэтому для освобождения памяти, занимаемой этими объектами в базе рисунка, нужно выполнить их удаление как примитивов с помощью функции `entdel`. Исключением из указанного правила являются объекты словарей групп и стилей мультилиний. Система AutoCAD сама освобождает память, занятую удаляемыми с помощью функции `dictremove` группами и стилями мультилиний.

Замечание

Система AutoCAD не дает возможность удалить с помощью функции `dictremove` стили мультилиний, занятые в объектах рисунка.

1.15.7. Функция *dictrename*

Переименовывает элемент словаря.

```
(dictrename <словарь> <старое> <новое>)
```

Аргументы: `<словарь>` — словарь как объект базы рисунка, `<старое>` и `<новое>` — текстовые строки (задают старое и новое имена переименовываемого элемента словаря).

Возвращаемое значение — значение аргумента `<новое>` при успешном переименовании. При ошибке возвращается `nil`.

Пример:

- `(dictrename (namedobjdict) "МОЙ_СЛОВАРЬ" "ТВОЙ_СЛОВАРЬ")` возвращает `"ТВОЙ_СЛОВАРЬ"` и переименовывает словарь, созданный в *разд. 1.15.5*.

1.15.8. Функция *layoutlist*

Получает имена вкладок листов текущего рисунка (в алфавитном порядке).

```
(layoutlist)
```

Возвращаемое значение — список строк (имя вкладки модели в список не входит).

Пример (см. файл Hummer Elevation.dwg из стандартных примеров, поставляемых вместе с системой AutoCAD 2006):

- `(layoutlist)` возвращает список с именами двух листов `("Color" "Mono")`

1.15.9. X-записи и наполнение словарей

X-записи — это особые неграфические объекты, которые могут создаваться пользователями и хранить произвольные данные. Как объекты (примитивы) они имеют тип с названием XRECORD.

X-записи являются аналогом расширенных данных *(см. разд. 1.14)*, но обладают преимуществом — их размер не ограничен 16383 байтами.

Создание X-записей выполняется с помощью функции `entmakex` и списков, содержащих, как и обычные примитивы, точечные пары с DXF-кодами меньше 1000, в отличие от расширенных данных, которые пользуются DXF-кодами 1000 и больше. Желательно ограничиваться DXF-кодами 1—369, поскольку отмечены случаи, когда система использует некоторые коды в диапазоне 370—999.

После создания X-записи группируются в словари пользователя. Рассмотрим этот процесс на примере, оформленном в виде листинга 1.24.

Листинг 1.24. Пример создания и наполнения пользовательского словаря

```
; Глава 01\Листинг 1.24.lsp
; Формирование списков данных, сохраняемых в X-записях
(setq lxr1 '((0 . "XRECORD") (100 . "AcDbXrecord")
  (1 . "Красная Москва") (10 110.0 5267.2 230.8)
  (11 -11.2 1234.6 429.0) (40 . 637.54) (70 . 0)))
(setq lxr2 (list '(0 . "XRECORD") '(100 . "AcDbXrecord")
  (cons 1 "Северная фантазия") (cons 10 (list 12.2 38.9 -2.4))
  (cons 11 (list 6.3 25.6 10.6)) (cons 40 30.5) (cons 70 1)))
; Создание X-записей
(setq xr1 (entmakex lxr1))
(setq xr2 (entmakex lxr2))
; Создание словаря КОСМЕТИКА
(dictadd (namedobjdict) "КОСМЕТИКА" (setq newdic
  (entmakex '((0 . "DICTIONARY") (100 . "AcDbDictionary")))))
; Добавление элементов к новому словарю
(dictadd newdic "Москва" xr1)
(dictadd newdic "Петербург" xr2)
```

Разберем пример и убедимся в его работоспособности. Сначала формируются списки `lxr1` и `lxr2` для будущих объектов типа XRECORD (точечные пары с DXF-кодами 0 и 100 должны быть стандартными, остальные задаются пользователем). Поскольку списки содержат только постоянные данные, то в первом списке для краткости использован апостроф. Второй список строится аналогично, но с применением функций list и cons (как это было бы обязательным для данных, извлекаемых из программных переменных).

Далее с помощью функции `entmakex` создаются объекты (X-записи) `xr1` и `xr2`:

`(setq xr1 (entmakex lxr1))` возвращает `<Entity name: 7ef66f78>`,

`(setq xr2 (entmakex lxr2))` возвращает `<Entity name: 7ef66f80>`.

Можем с помощью функции `entget` проверить данные построенных X-записей:

- `(entget xr1)` возвращает `((-1 . <Entity name: 7ef66f78>) (0 . "XRECORD") (330 . <Entity name: 0>) (5 . "E7") (100 . "AcDbXrecord") (280 . 1) (1 . "Красная Москва") (10 110.0 5267.2 230.8) (11 -11.2 1234.6 429.0) (40 . 637.54) (70 . 0))`
- `(entget xr2)` возвращает `((-1 . <Entity name: 7ef66f80>) (0 . "XRECORD") (330 . <Entity name: 0>) (5 . "E8") (100 . "AcDbXrecord") (280 . 1) (1 . "Северная фантазия") (10 12.2 38.9 -2.4) (11 6.3 25.6 10.6) (40 . 30.5) (70 . 1))`

Видно, что интерпретатор LISP при выполнении функции `enmakex` дополнил первоначальные данные X-записей точечными парами с DXF-кодами 330, 5 и 280.

Новый словарь КОСМЕТИКА добавляется к корневому словарю точно так же, как мы это сделали в *разд. 1.15.5*. Проверим, что в списке словарей появился словарь КОСМЕТИКА:

- `(entget (namedobjdict))` возвращает `((-1 . <Entity name: 7ef66c60>) (0 . "DICTIONARY") (330 . <Entity name: 0>) (5 . "C") (100 . "AcDbDictionary") (280 . 0) (281 . 1) (3 . "ACAD_COLOR") (350 . <Entity name: 7ef66dd8>) (3 . "ACAD_GROUP") (350 . <Entity name: 7ef66c68>) (3 . "ACAD_LAYOUT") (350 . <Entity name: 7ef66cd0>) (3 . "ACAD_MATERIAL") (350 . <Entity name: 7ef66dd0>) (3 . "ACAD_MLINESTYLE") (350 . <Entity name: 7ef66cb8>) (3 . "ACAD_PLOTSETTINGS") (350 . <Entity name: 7ef66cc8>) (3 . "ACAD_PLOTSTYLENAME") (350 . <Entity name: 7ef66c70>) (3 . "ACAD_TABLESTYLE") (350 . <Entity name: 7ef66e30>) (3 . "AcDbVariableDictionary") (350 . <Entity name: 7ef66d70>) (3 . "КОСМЕТИКА") (350 . <Entity name: 7ef66f88>) (3 . "ТВОЙ_СЛОВАРЬ") (350 . <Entity name: 7ef66f68>))`

В конце примера в словарь КОСМЕТИКА заносятся объекты `xr1` и `xr2` с именами `Москва` и `Петербург`. Если проверить с помощью функции `entget` списки данных X-записей `xr1` и `xr2`, то мы увидим, что списки изменились (появилась связь со словарем):

- `(entget xr1)` возвращает `((-1 . <Entity name: 7ef66f78>) (0 . "XRECORD") (5 . "E7") (102 . "{ACAD_REACTORS") (330 . <Entity name: 7ef66f88>) (102 . "}") (330 . <Entity name: 7ef66f88>) (100 . "AcDbXrecord") (280 . 1) (1 . "Красная Москва") (10 110.0 5267.2 230.8) (11 -11.2 1234.6 429.0) (40 . 637.54) (70 . 0))`
- `(entget xr2)` возвращает `((-1 . <Entity name: 7ef66f80>) (0 . "XRECORD") (5 . "E8") (102 . "{ACAD_REACTORS") (330 . <Entity name: 7ef66f88>) (102 . "}") (330 . <Entity name: 7ef66f88>) (100 . "AcDbXrecord") (280 . 1) (1 . "Северная фантазия") (10 12.2 38.9 -2.4) (11 6.3 25.6 10.6) (40 . 30.5) (70 . 1))`

Проверим с помощью функции `dictnext`, какие элементы имеются в словаре КОСМЕТИКА:

- `(dictnext newdic T)` возвращает `((-1 . <Entity name: 7ef66f78>) (0 . "XRECORD") (5 . "E7") (102 . "{ACAD_REACTORS") (330 . <Entity name: 7ef66f88>) (102 . "}") (330 . <Entity name: 7ef66f88>)`

```
(100 . "AcDbXrecord") (280 . 1) (1 . "Красная Москва")
(10 110.0 5267.2 230.8) (11 -11.2 1234.6 429.0) (40 . 637.54) (70 . 0))
```

- `(dictnext newdic)` возвращает `((-1 . <Entity name: 7ef66f80>) (0 . "XRECORD") (5 . "E8") (102 . "{ACAD_REACTORS") (330 . <Entity name: 7ef66f88>) (102 . "}") (330 . <Entity name: 7ef66f88>) (100 . "AcDbXrecord") (280 . 1) (1 . "Северная фантазия") (10 12.2 38.9 -2.4) (11 6.3 25.6 10.6) (40 . 30.5) (70 . 1))`

Аналогичного результата также можно было бы добиться с помощью выражений `(dictsearch newdic "Москва")` и `(dictsearch newdic "Петербург")`.

Таким образом, в словаре КОСМЕТИКА имеются два объекта. Как использовать данные этих объектов, зависит от прикладного программного обеспечения.

1.16. Разработка функций пользователя

Пользователь может создавать свои функции и применять их параллельно со встроенными. Этой цели служат LISP-функция `lambda`, используемая для описания безымянных одноразовых функций, и `defun`, используемая для определения новых именованных функций. Обе функции являются традиционными инструментами разработки функций пользователя в системе AutoCAD. Примеры пользовательских функций различного назначения с подробными комментариями и описаниями переменных приведены в приложениях.

Не рекомендуется в качестве имен своих функций брать зарезервированные имена, т. к. это приведет к некорректной работе отдельных команд системы AutoCAD. Для того чтобы получить полный перечень зарезервированных символов LISP, можно воспользоваться функцией `atoms-family` *(см. разд. 1.16.16)*.

Помимо основных средств разработки функций пользователя далее рассматриваются также функции, которые дают дополнительные возможности для программирования таких операций, как досрочный выход из программы, анализ ошибок, трассировка символов и т. п.

1.16.1. Функция *lambda*

Функция `lambda` позволяет создавать безымянные пользовательские функции (одноразового применения), которые можно определить и тут же выполнить в любом месте программы с нужными значениями аргументов. Такие функции нужны, например, когда некоторые выражения являются элементами сложных формул одноразового использования. Тогда эти выражения оформляются в качестве аргументов, а формулы — в качестве тела функции `lambda`.

Формат функции:

```
(lambda <аргументы> <выражения>)
```

Параметр `<аргументы>` является списком временных переменных, а параметр `<выражения>` — любой последовательностью выражений, оперирующих выбранными временными переменными.

Возвращается результат последнего вычисленного выражения из заданных в аргументе `<выражения>`.

Обращение к описанной в программе функции `lambda` осуществляется с помощью функций `apply` или `mapcar`. Поясним на примерах.

Выражение `(lambda (u v w) (abs (+ u (* 2 v) (* 3 w))))` является описанием безымянной функции, вычисляющей абсолютную величину суммы `u+2*v+3*w`, где `u`, `v` и `w` — имена временных переменных, выбранных для обозначения аргументов (действие переменных не распространяется за пределы функции `lambda`). Для использования созданной таким образом одноразовой функции можно в программе вставить, например, текст:

- `(apply '(lambda (u v w) (abs (+ u (* 2 v) (* 3 w)))) '(4.7 -3.2 10.5))`

  результатом которого будет `29.8`. При вычислении AutoLISP подставит вместо `u`, `v` и `w`, соответственно, `4.7`, `-3.2` и `10.5` и вычислит выражение `(abs (+ 4.7 (* 2 -3.2) (* 3 10.5)))`.

Второй пример:

- `(mapcar '(lambda (q) (+ 1.0 q (* q q) (* q q q)))`
  `(list 1.7 2.2 5.8 3.0))` возвращает `(10.503 18.688 235.552 40.0)`

  Здесь функция, вычисляющая выражение $1+q+q^2+q^3$, поочередно применяется к каждому из элементов списка `(list 1.7 2.2 5.8 3.0)`.

1.16.2. Функция *defun*

Для создания функций пользователя предназначена специальная функция `defun`:

```
(defun <имя> ([<аргументы>] [/ <рабочие>]) <выражения>)
```

Описывает тело функции, ее аргументы и рабочие переменные (т. е. переменные, которые не являются глобальными и имеют действие только внутри тела самой функции).

Аргументы: `<имя>` — символ, используемый в качестве имени новой функции; `<аргументы>` — символы, разделяемые пробелами и используемые в качестве аргументов новой функции; `<рабочие>` — символы, разделяемые пробелами, которые используются в качестве временных (рабочих) переменных новой функции; `<выражения>` — любая последовательность выражений, использующих как аргументы и рабочие переменные, так и другие функции и глобальные переменные.

Возвращаемым значением самой функции `defun` является символ, указанный в качестве аргумента `<имя>`.

При обращении к пользовательской функции, созданной с помощью `defun`, указываются только имя функции и значения ее аргументов, причем в качестве каждого из аргументов могут передаваться выражения. Возвращаемым значением новой функции является результат последнего вычисленного выражения ее тела с переданными значениями аргументов.

Пример простой функции приведен в листинге 1.25.

Листинг 1.25. Пример пользовательской функции `oursum`

```
; Глава 01\Листинг 1.25.lsp
(defun oursum (a1 a2 a3 / s)
  (setq s (+ a1 a2))
  (- s a3)
); конец тела функции oursum
```

Здесь определена новая функция с именем `oursum`. У функции три аргумента (`a1`, `a2` и `a3`) и одна рабочая переменная (`s`). Переменная `s` используется для временного хранения результата операции суммирования. В теле функции всего два выражения: первое вычисляет сумму `a1` и `a2` и сохраняет ее в переменной `s`, а второе рассчитывает разность между `s` и `a3`. Вычисленная разность представляет собой значение последнего выражения в теле функции, поэтому является и возвращаемым значением функции `oursum`. Как аргументы `a1`, `a2` и `a3`, так и переменная `s` называются *локальными переменными*, т. е. их значения не являются глобальными и не доступны вне тела функции.

Для того чтобы функцией `oursum` можно было воспользоваться в текущем рисунке AutoCAD, необходимо либо ввести ее описание в командной строке AutoCAD (что чаще всего довольно трудоемко), либо предварительно сохранить описание в текстовом файле с расширением lsp (например, D:\Temp\ourfunctions.lsp) и затем загрузить этот файл с помощью функции `load`:

```
(load "D:\\Temp\\ourfunctions.lsp")
```

Если имя загружаемого файла с расширением lsp не содержит пути, то система AutoCAD ищет файл в путях доступа к файлам поддержки, заданных в настройке. После того как файл, содержащий тело функции `oursum`, загружен, можно обратиться к функции в любое время текущего сеанса редактирования и в любом месте программы, задав значения ее трех аргументов, например:

```
(setq newval1 (oursum 55.02 (sqrt 0.933) -2.64))
```

Разумеется, при обращении к функции `oursum` в качестве аргументов могут быть использованы любые выражения, результатами вычисления которых будут значения необходимого типа (в данном примере — это три числа).

Замечание

Файл с расширением lsp не обязан содержать только тело одной функции. В нем могут присутствовать тела других функций или исполняемые выражения.

Загрузить файл с LISP-программами можно и с помощью диалогового окна **Load/Unload Applications** (Загрузка/выгрузка приложений), вызываемого с помощью пункта меню **Tools | Load Application** (Сервис | Приложения).

Вспомним пример программы работы с примитивами рисунка, приведенный в листинге 1.18 *разд. 1.12.23*. Теперь эту программу можно оформить в качестве функции с наименованием `rcl` (листинг 1.26). В конце функции добавим тихий выход.

Листинг 1.26. Пример функции `rcl`, работающей с примитивами рисунка

```
; Глава 01\Листинг 1.26.lsp
(defun rcl ( / )
; Начальные значения списков радиусов (list_rad) и центров (list_cen)
(setq list_rad '() list_cen '())
; Создание набора из кругов на слое HOLES
(setq nab_cir (ssget "_X" (list (cons 8 "HOLES") (cons 0 "CIRCLE"))))
; Проверка, сформировался ли набор nab_cir
; (если нет, то предыдущая операция вернет nil)
(if (null nab_cir)
  (progn
    (princ "\nНет кругов на слое HOLES. "); сообщение об отсутствии
    (princ); тихий выход
  ); конец progn
  (progn
    (setq i -1 nab_len (sslength nab_cir))
; Цикл по количеству элементов набора nab_cir
    (repeat nab_len
      (setq i (1+ i))
; Выбор следующего примитива и получение его списка
      (setq cirlist (entget (ssname nab_cir i)))
      (setq radcir (cdr (assoc 40 cirlist)))
      (setq cencir (cdr (assoc 10 cirlist)))
; Добавление радиуса и точки центра к спискам list_rad и list_cen
      (setq list_rad (append list_rad (list radcir)))
      (setq list_cen (append list_cen (list cencir)))
   ); конец repeat
; Печать результирующих списков
    (princ "\nРадиусы: ")
    (princ list_rad)
    (princ "\nЦентры: ")
    (princ list_cen)
 ); конец progn
); конец if
(princ); тихий выход
); конец defun
```

Программа оформлена в виде пользовательской функции `rcl`. Функцию `rcl` следует сохранить (например, в файле D:\Temp\rcl.lsp), после чего ее можно загрузить в память:

```
(load "D:\\Temp\\rcl.lsp")
```

Если файл загрузился без ошибок, то система AutoCAD должна вернуть `RCL` — имя последней загруженной функции, переведенное в верхний регистр. Теперь остается только обратиться к ней в командной строке:

```
(rcl)
```

Возможный вид результата работы функции `rcl`:

```
Радиусы: (66.0916 29.1292)
Центры: ((179.603 194.961 0.0) (43.0525 184.539 0.0))
```

В данном примере все использованные в функции `rcl` переменные были глобальными, что позволит воспользоваться их значениями другим программам. Однако нетрудно заметить, что по окончании работы функции `rcl` полезными являются только переменные `list_rad` и `list_cen`. Остальные переменные хранят промежуточные результаты и скорее всего уже больше не нужны. Для аккуратности следует либо всем ненужным переменным в конце работы программы присвоить значение `nil`, либо включить их имена в список рабочих переменных функции `rcl`, например:

```
(defun rcl ( / nab_cir i nab_len radcir cencir cirlist) ...
```

После того как ваша функция загружена в оперативную память, она вносится в список LISP-символов данного сеанса редактирования системы AutoCAD. Поэтому значение этого символа можно прочитать с помощью операции `!` (восклицательный знак):

- `!rcl` возвращает `#<USUBR @0ab1f4c4 RCL>` (конкретные цифры функции, естественно, могут у вас быть другими)

1.16.3. Создание дополнительных команд системы AutoCAD с помощью пользовательских функций

Особое значение имеют пользовательские функции, имена которых содержат префикс `c:`, а сами функции не имеют аргументов. Такие функции не только являются функциями, к которым можно обратиться из программы на языке AutoLISP, но они еще и определяют новые команды системы AutoCAD (имя команды получается из имени функции отбрасыванием префикса).

Пример создания команды PRITENT приведен в листинге 1.27.

Листинг 1.27. Пример создания команды PRITENT системы AutoCAD

```
; Глава 01\Листинг 1.27.lsp
(defun c:pritent ( / wn)
  (setq wn (entsel "\nУкажите объект для определения его типа: "))
```

```
  (if (null wn)
    (princ "Объект не выбран. ")
    (progn
      (princ "Тип объекта: ")
      (princ (cdr (assoc 0 (entget (car wn)))))
    );progn
  );if
  (princ); тихий выход
);defun c:pritent
```

После загрузки такая функция работает не только как функция, к которой можно обратиться с помощью выражения `(c:pritent)`, но и как команда AutoCAD с именем PRITENT, которое можно вводить в командной строке. На рис. 1.13 приведен пример работы только что созданной команды.

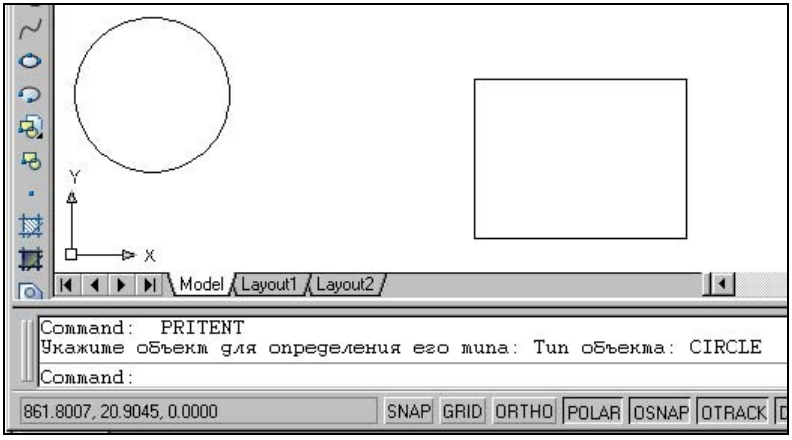


Рис. 1.13. Пример работы новой команды PRITENT

О выводе имен функций, формирующих дополнительные команды системы AutoCAD, см. функцию `vl-list-exported-functions` *(разд. 1.17.18)*.

1.16.4. Функция *defun-q*

Создает определение пользовательской функции аналогично функции `defun`, но при чтении значения символа с именем функции возвращается список с ее телом.

```
(defun-q <имя> ([<аргументы>] [/ <рабочие>]) <выражения>)
```

Аргументы — те же, что и у функции `defun`.

Возвращаемым значением самой функции `defun-q` является символ, указанный в качестве первого аргумента.

Функция defun-q введена как временный инструмент для совместимости с ранними версиями системы AutoCAD и в следующих версиях может не поддерживаться.

Рассмотрим отличия между функциями defun и defun-q на примере пользовательской функции oursum, приведенной в *разд. 1.16.2*. Создадим аналогичную функцию oursumq, но с помощью defun-q:

```
(defun-q oursumq (a1 a2 a3 / s) (setq s (+ a1 a2)) (- s a3))
```

Если загрузить обе функции — oursum и oursumq — в память с помощью функции load, то обе функции будут работать одинаково:

- (oursum 15 18 6) возвращает 27
- (oursumq 15 18 6) возвращает 27

Однако если посмотреть значения символов oursum и oursumq, то получим разные результаты:

- !oursum возвращает #<USUBR @0acc1a28 OURSUM> (имя указателя функции)
- !oursumq возвращает список ((A1 A2 A3 / S) (SETQ S (+ A1 A2)) (- S A3))

В последнем списке видим тело функции, без ее имени. Первый элемент списка является списком символов аргументов и символов рабочих переменных, а остальные элементы списка — исполняемые выражения, составляющие суть функции. К этому списку можно обращаться как к обычному списку:

- (car oursumq) возвращает (A1 A2 A3 / S)
- (cdr oursumq) возвращает ((SETQ S (+ A1 A2)) (- S A3))

Поэтому можно вносить изменения в тело функции oursumq с помощью операций редактирования списка. Можно создавать новые функции, используя тело функции oursumq как основу, например:

- (setq oursumq1 (list (car oursumq) (cadr oursumq) '(* s a3))) возвращает ((A1 A2 A3 / S) (SETQ S (+ A1 A2)) (* S A3))

Таким образом, мы создали и загрузили новую функцию oursumq1, которая отличается от oursumq тем, что последняя операция вычитания (- S A3) заменена в ней на операцию умножения (* S A3). Поэтому выражение (oursumq1 15 18 6) возвратит уже 198, а не 27.

1.16.5. Функция *defun-q-list-ref*

Получает список тела функции, ранее созданной с помощью defun-q (фактически выполняет операцию, аналогичную восклицательному знаку в командной строке).

```
(defun-q-list-ref '<функция-q>)
```

Аргумент — символ с именем функции, созданной с помощью функции defun-q.

Возвращаемое значение — список с телом функции. Возвращается `nil`, если в качестве аргумента подано либо не имя функции, либо имя функции, созданной не с помощью функции `defun-q`.

Примеры (продолжение примеров из *разд. 1.16.4*):

- `(defun-q-list-ref 'oursumq1)` возвращает `((A1 A2 A3 / S) (SETQ S (+ A1 A2)) (* S A3))`
- `(defun-q-list-ref 'oursumq)` возвращает `((A1 A2 A3 / S) (SETQ S (+ A1 A2)) (- S A3))`
- `(defun-q-list-ref 'oursum)` возвращает `nil`

1.16.6. Функция *defun-q-list-set*

Создает функцию из списка по методу, аналогичному методу функции `defun-q`.

```
(defun-q-list-set '<функция-q> <список>)
```

Аргументы: `<функция-q>` — символ, используемый в качестве имени создаваемой функции; `<список>` — список с телом функции (как правило, его тоже нужно предварять апострофом, чтобы он не вычислялся раньше времени).

Возвращаемое значение — символ с именем создаваемой функции.

Пример:

- `(defun-q-list-set 'yourf '((w) (sin (+ w w))))` создает функцию и возвращает `YOURF`

1.16.7. Функция *function*

Сообщает компилятору Visual LISP (о компиляции *см. главу 2*) о необходимости связывания и оптимизации функции, имя которой указано в качестве первого аргумента, или оптимизации выражения с функцией `lambda`, использованного в качестве значения последнего аргумента (обычные безымянные пользовательские функции, создаваемые с помощью функции `lambda`, не участвуют в оптимизации исполняемого кода программы).

```
(function <параметр>)
```

Аргумент `<параметр>` — символ, используемый в качестве имени функции, или выражение, начинающееся с функции `lambda`.

Пример:

- `(function (lambda (g) (sqrt (+ g 56.9))))` возвращает `(quote #<SUBR @01e27ed8 -lambda->)`

Функция `function` позволяет заменить апостроф перед символом с именем функции (как в первом аргументе функции `mapcar`). Например:

- `(mapcar (function (lambda (g) (sqrt (+ g 56.9)))) (list 1.0 2.0))` возвращает `(7.6092 7.67463)`

1.16.8. Функция *vlisp-compile*

Пользовательские файлы с LISP-программами можно компилировать. Для этого предназначена специальная функция `vlisp-compile`:

```
(vlisp-compile '<режим> <имя1> [<имя2>])
```

Функция имеет следующие аргументы:

- `<режим>` — наименование режима компиляции; может принимать значения `st` (стандартный), `lsm` (оптимизация и непрямое связывание) или `lsa` (оптимизация и прямое связывание); аргументу должен обязательно предшествовать апостроф;
- `<имя1>` — имя компилируемого файла с расширением lsp;
- `<имя2>` — имя выходного файла с расширением fas.

Обычное значение параметра `<режим>` — `st`. При необходимости оптимизировать исполняемый код можно воспользоваться другими вариантами.

В результате компиляции образуется файл с расширением fas, на загрузку которого требуется меньше времени, чем на загрузку файла с исходным кодом (с расширением lsp), поэтому для отлаженных программ использование FAS-файлов предпочтительнее. Для загрузки компилированных файлов также применяется функция `load`.

Если аргумент `<имя2>` не задан, то имя компилированного файла получается из имени компилируемого файла отбрасыванием расширения lsp и добавлением расширения fas.

Возвращаемое значение — `T`, если компиляция закончилась успешно, и `nil` — при ошибке.

Замечание

Функция компиляции работает только при загруженной среде разработки приложений Visual LISP *(см. главу 2)*.

1.16.9. Функция *exit*

Вызывает принудительный выход из приложения.

```
(exit)
```

Возвращаемого значения нет. В командную строку выводится сообщение об ошибке следующего вида:

; error: quit / exit abort

(; ошибка: завершить / выйти прервать)

1.16.10. Функция *quit*

Так же, как и предыдущая функция, вызывает принудительный выход из приложения.

```
(quit)
```

Возвращаемого значения нет. В командную строку выводится сообщение:

; error: quit / exit abort

(; ошибка: завершить / выйти прервать)

1.16.11. Функция **error**

Получает в сбойных ситуациях от системы AutoCAD строку с некоторым сообщением и обрабатывает ее (обычно выводит разъясняющее сообщение).

```
(*error* <сообщение>)
```

Данную функцию могут изменять в своих программах пользователи, чтобы выводить собственные варианты сообщений. Поэтому рекомендуется перед тем, как дать свое содержание функции `*error*`, запомнить ее предыдущее состояние (для последующего восстановления).

Аргумент — текстовая строка.

Возвращаемого значения нет (кроме особых случаев — см. описание функции `vl-exit-with-value`). В теле функции `*error*` должна присутствовать какая-либо операция вывода сообщения на экран.

Пример приведен в листинге 1.28.

Листинг 1.28. Пример пользовательской функции `*error*`

```
; Глава 01\Листинг 1.28.lsp
; Сохранение стандартной функции обработки ошибок
(setq olderror *error*)
; Тело новой функции, вызываемой вместо стандартной
(defun *error* (msg) (princ "\nОшибка: ")
  (if
    (or
      (= msg "завершить / выйти прервать")
      (= msg "quit / exit abort")
   ); or
    (princ "принудительный выход из приложения ")
    (princ msg)
 ); if
  (princ)
); defun
```

```
;; В дальнейшем восстановить стандартную функцию:
; (setq *error* olderror)
```

В первой строке листинга запоминаем в переменной `olderror` предыдущее состояние функции `*error*` (возвращается `#<SUBR @099073fc *ERROR*>`).

После загрузки функции `*error*` с новым телом система будет работать с новым обработчиком сбойных ситуаций. Например, при выходе из программы с помощью функции `exit` пользователь получит не стандартное сообщение `quit / exit abort`, а следующее сообщение:

```
Ошибка: принудительный выход из приложения
```

Затем можно восстановить предыдущее состояние функции `*error*`:

```
(setq *error* olderror)
```

См. также функции `vl-exit-with-error`, `vl-exit-with-value`, `vl-catch-all-apply`, `vl-catch-all-error-message` и `vl-catch-all-error-p`.

1.16.12. Функция *vl-acad-defun*

Делает функцию с именем, имеющим префикс `c:`, доступной приложениям ObjectARX.

```
(vl-acad-defun '<функция>)
```

Аргумент — имя функции на языке LISP.

Возвращаемое значение не определено.

Пример:

- `(vl-acad-defun 'c:pritent)` дает возможность приложениям, написанным с помощью ObjectARX, обращаться к функции `c:pritent`

1.16.13. Функция *vl-acad-undefun*

Выполняет обратное, по отношению к функции `vl-acad-defun`, действие.

```
(vl-acad-undefun '<функция>)
```

Аргумент — имя функции на языке LISP.

Возвращаемое значение — `T`, если операция выполнена успешно, и `nil` — при ошибке (например, если имя функции указано неверно).

Пример:

- `(vl-acad-undefun 'c:pritent)` не дает возможности ARX-приложениям обращаться к функции `c:pritent`

1.16.14. Функция *vl-exit-with-error*

Позволяет VLX-приложению, работающему в собственном пространстве имен *(см. разд. 1.17.32)*, вызывать для обработки ошибок функцию `*error*` и передавать управление командной строке.

```
(vl-exit-with-error <сообщение>)
```

Аргумент — текстовая строка.

Возвращаемого значения нет.

Обращение к функции `vl-exit-with-error` должно быть включено в переопределение функции `*error*`, например:

```
♦ (defun *error* (msg) (princ "Нарушение нормальной работы. ")
  (vl-exit-with-error (strcat "Приложение блокировано: " msg))
  ); defun
```

Тогда в случае сбоя работы VLX-приложения управление будет передано в пользовательскую функцию обработки ошибок вызывающего пространства имен, с генерацией соответствующего сообщения.

1.16.15. Функция *vl-exit-with-value*

Позволяет VLX-приложению в случае сбойной ситуации вернуть в другое пространство имен то же значение, что было передано фунции `vl-exit-with-value` в качестве ее аргумента.

```
(vl-exit-with-value <значение>)
```

Аргумент — любое значение (чаще всего — целое число). Это же значение становится возвращаемым значением функции и VLX-приложения в случае передачи управления программе обработки ошибок.

Пример:

```
♦ (defun *error* (msg) (princ "Нарушение нормальной работы. ")
  (vl-exit-with-value 517))
```

В случае сбоя VLX-приложение вернет значение `517`.

1.16.16. Функция *atoms-family*

При создании своих функций не рекомендуется в качестве имен функций выбирать зарезервированные имена, т. к. это приведет к некорректной работе отдельных команд системы AutoCAD. Для того чтобы получить полный перечень зарезервированных символов LISP, можно воспользоваться функцией `atoms-family`, имеющей следующий синтаксис:

```
(atoms-family <форма> [<список>])
```

Аргументы: <форма> — целое число, может принимать значения только 0 или 1; <список> — список из текстовых строк (имен, проверяемых на резервирование).

Возвращаемое значение — список зарезервированных имен символов LISP. Если значение аргумента <форма> равно 0, то элементами возвращаемого списка будут символы, а если 1 — то строки. Если аргумент <список> не задан, то возвращается полный список зарезервированных символов, если задан — то возвращается та часть аргумента <список>, которая содержит используемые зарезервированные имена и nil для незанятых символов из числа проверяемых.

Пример:

- (acad_strlsort (atoms-family 1)) возвратит список зарезервированных символов, который будет тут же с помощью функции acad_strlsort отсортирован по алфавиту

1.16.17. Функция *trace*

Трассирует значения заданных функций.

```
(trace [<функция1> [<функция2> ... [<функцияN>] ... ]])
```

Аргументы — символы с именами функций.

Возвращаемое значение — имя функции, заданной в качестве последнего аргумента функции. Если аргументы не заданы, то возвращается nil.

Если загружена среда Visual LISP, то значения трассируемых функций выводятся в окно **Trace** (Трассировка) *(об отладке см. разд. 2.4.1)*. Если среда Visual LISP не загружена, тогда значения трассируемых функций выводятся в командную строку системы AutoCAD.

Отмена режима трассировки выполняется рассматриваемой ниже функцией untrace.

Пример (на базе функции oursum из *разд. 1.16.2*):

- (trace oursum) возвращает OURSUM

Ввод выражения (oursum 3 7 20) возвратит следующие строки (приводятся варианты английской и русской версий системы AutoCAD):

Entering (OURSUM 3 7 20)
(Ввод (OURSUM 3 7 20))
Result: -10
(Результат: -10)
-10

Если функция oursum будет использована во вводимых выражениях большее количество раз, то оформление трассировки будет похожим, например:

- `(+ (oursum 15 3 0) (oursum 2 56 4) (oursum 0 1 2))` возвратит строки трассировки три раза

1.16.18. Функция *untrace*

Отменяет трассировку заданных функций (см. функцию `trace`).

```
(untrace [<функция1> [<функция2> ... [<функцияN>] ... ]])
```

Аргументы — символы с именами функций.

Возвращаемое значение — имя функции, заданной в качестве последнего аргумента функции. Если аргументы не заданы, то возвращается `nil`.

Пример:

- `(untrace oursum)` возвращает `OURSUM`

1.17. Функции работы с памятью, приложениями и реестром Windows

Далее описываются различные функции, выполняющие системные операции (в основном со средой, в которой функционируют разрабатываемые приложения). При знакомстве с функциями *разд. 1.17* могут понадобиться некоторые сведения об ARX-приложениях (на языке C++) и VLX-приложениях. Вопросы разработки ARX-приложений в настоящей книге не затрагиваются[6]. О создании VLX-приложений *см. главу 2*.

В конце *разд. 1.17* освещаются вопросы работы с переменными и функциями пользователя, пространствами имен и многодокументным режимом.

1.17.1. Функция *ver*

Возвращает строку с наименованием версии Visual LISP.

```
(ver)
```

Пример (в русской и английской версиях системы AutoCAD 2006 дает одинаковый результат):

- `(ver)` возвращает `"Visual LISP 2006 (en)"`

1.17.2. Функция *mem*

Выводит статистику распределения странично организованной памяти и статистику операций сборки мусора (см. функцию `gc`).

```
(mem)
```

[6] См. книгу "AutoCAD: разработка приложений, настройка и адаптация".

Возвращаемое значение — `nil`.

Пример:

- `(mem)` выводит:

```
; GC calls: 19; GC run time: 16 ms
Dynamic memory segments statistic:
PgSz  Used  Free  FMCL  Segs  Type
 128     4   507   505     1  bstack body
4096   365    10     9    25  DMxx memory
 512     1   126   126     1  undo strings
4096   147   138    14    19  DM Str
  32  1028   955   899     1  ::new
4096   478     2     2    32  CONS memory
 256  2329  2261   209    18  bytecode area
 512    39   215   109     2  lisp stacks
Segment size: 65536, total used: 99, free: 0
nil
```

1.17.3. Функция *alloc*

Устанавливает размер сегмента для последующего вызова функции `expand`.

```
(alloc <размер>)
```

Аргумент — целое число (количество размещаемых в сегменте символов, строк, функций, вещественных чисел и т. д.).

Возвращаемое значение — целое число (предыдущее значение размера сегмента).

Пример:

- `(alloc 120)` возвращает `256`

1.17.4. Функция *expand*

Отводит в памяти дополнительное количество сегментов для работы LISP (размер сегмента устанавливается функцией `alloc`).

```
(expand <количество>)
```

Аргумент — положительное целое число.

Возвращаемое значение — целое число (размер дополнительно отведенной памяти для хранения данных LISP).

Пример:

- `(alloc 100)` возвращает `1000`
- `(expand 2)` возвращает `82`

Это означает, что AutoLISP получает дополнительно место для размещения в памяти 200 символов (строк, функций, вещественных чисел) или 8200 точечных пар.

1.17.5. Функция *gc*

Выполняет сборку мусора (освобождает память от неиспользуемых данных).

```
(gc)
```

Возвращаемое значение — `nil`.

Когда работают большие приложения, то может возникнуть необходимость высвобождения памяти в куче (heap), поскольку не хватает места для размещения очередной порции временных данных. В такой ситуации операция сборки мусора может принести пользу.

1.17.6. Функция *vl-registry-read*

Читает данные параметра раздела реестра Windows.

```
(vl-registry-read <раздел> [<параметр>])
```

Аргументы — текстовые строки с именами раздела и параметра реестра Windows.

Возвращаемое значение — данные (значение) параметра из указанного раздела реестра. Если аргумент `<параметр>` не задан, то возвращаются значения всех параметров данного раздела.

Примеры:

- `(vl-registry-read "HKEY_CURRENT_USER\\Control Panel\\Colors" "ButtonLight")` возвращает `"192 192 192"`
- `(vl-registry-read "HKEY_CURRENT_USER\\SOFTWARE\\Autodesk\\AutoCAD \\R16.2\\ACAD-4001:409\\MTEXT" "AutoStack")` возвращает `1`

1.17.7. Функция *vl-registry-write*

Создает раздел реестра или записывает туда параметр и его значение.

```
(vl-registry-write <раздел> [<параметр> [<значение>]])
```

Все аргументы строковые: `<раздел>` — имя раздела реестра, `<параметр>` — имя параметра раздела, `<значение>` — значение, присваиваемое параметру с заданным именем.

Возвращается значение аргумента `<значение>`. При ошибке возвращается `nil`.

Данная функция не может записывать в разделы, начинающиеся с `HKEY_USERS` или `HKEY_LOCAL_MACHINE`.

Если значение аргумента <параметр> опущено или равно nil, то запись выполняется в параметр, действующий по умолчанию (его имя — пустая строка). Если аргумент <значение> не задан, то в качестве значения записывается пустая строка.

Пример:

- (vl-registry-write "HKEY_CURRENT_USER\\New3" "" "1234567890") возвращает "1234567890"

1.17.8. Функция *vl-registry-delete*

Удаляет параметр из раздела реестра или целый раздел.

(vl-registry-delete <раздел> [<параметр>])

Все аргументы строковые: <раздел> — имя раздела реестра, <параметр> — имя параметра раздела.

Возвращаемое значение — T, если удаление прошло успешно. При ошибке возвращается nil.

Если аргумент <параметр> не задан или задан равным nil, то удаляются все параметры из раздела с именем <раздел>.

Пример:

- (vl-registry-delete "HKEY_CURRENT_USER\\New3") возвращает T, если удаление прошло успешно

1.17.9. Функция *vl-registry-descendents*

Выводит список подразделов данного раздела или список его параметров.

(vl-registry-descendents <раздел> [<признак>])

Аргументы: <раздел> — текстовая строка (имя раздела реестра); <признак> — любое значение или nil.

Если значение аргумента <признак> не задано или задано равным nil, то выводятся имена всех подразделов указанного раздела. В других случаях выводятся имена параметров, находящихся в данном разделе.

Возвращаемое значение — список из строковых значений или nil.

Примеры:

- (vl-registry-descendents "HKEY_CURRENT_USER\\Control Panel\\Colors") возвращает nil
- (vl-registry-descendents "HKEY_CURRENT_USER\\Control Panel\\Colors" "") возвращает ("MenuBar" "MenuHilight" "WindowText" "WindowFrame" "Window" "TitleText" "Scrollbar" "MenuText" "Menu" "InfoWindow" "InfoText" "InactiveTitleText" "InactiveTitle" "InactiveBorder" "HotTrackingColor" "HilightText" "Hilight" "GrayText" "GradientInactiveTitle"

```
"GradientActiveTitle" "ButtonText" "ButtonShadow" "ButtonLight"
"ButtonHilight" "ButtonFace" "ButtonDkShadow" "ButtonAlternateFace"
"Background" "AppWorkSpace" "ActiveTitle" "ActiveBorder")
```

1.17.10. Функция *arx*

Выводит список всех загруженных в данный момент файлов ARX-приложений.

```
(arx)
```

Возвращаемое значение — список из строковых величин.

Пример:

- `(arx)` возвращает `("acapp.arx" "acdblclkedit.arx" "acdblclkeditpe.arx" "acdim.arx" "acdwgrecovery.arx" "acdxnotify.arx" "aceplotx.arx" "acetlodr.arx" "acetutil.arx" "achapi16.dbx" "achlnkui.arx" "acidropmgr.arx" "aclaunchnfw.arx" "aclayerp.arx" "acmgd.dll" "acprevinput.arx" "acsign.arx" "acsmdwgmine.arx" "acsmnav.arx" "acsmpublishutils.arx" "acspacetrans.arx" "acstd.arx" "actaskbar.arx" "actp.arx" "fscreen.arx" "movebak.arx" "oleaprot.arx" "vl.arx" "whohas.arx" "wscommcntraccon.arx")`

1.17.11. Функция *arxload*

Загружает файл с ARX-приложением. При сбое загрузки выполняет заданные операции.

```
(arxload <файл> [<сбой>])
```

Аргументы: `<файл>` — текстовая строка с именем загружаемого файла (расширение arx в имени файла может быть опущено); `<сбой>` — любое выражение, которое будет вычислено (выполнено) при невозможности загрузки файла.

Возвращаемое значение — строка с именем файла, при успешной загрузке. При сбое загрузки возвращается результат вычисления выражения, заданного на случай ошибки в аргументе `<сбой>` (если аргумент `<сбой>` не был задан, то не возвращается ничего, но выводится стандартное сообщение об ошибке).

Примеры:

- `(arxload "geomcal.arx")` возвращает `"geomcal.arx"`
- `(arxload "noarx" (setq na -5))` возвращает `-5`, если файл noarx.arx не может быть загружен (при этом в результате выполнения выражения, записанного в качестве второго аргумента, переменной `na` присваивается `-5`)

1.17.12. Функция *arxunload*

Выгружает файл с ARX-приложением. При сбое выгрузки выполняет заданные операции.

```
(arxunload <файл> [<сбой>])
```

Аргументы: <файл> — текстовая строка с именем выгружаемого файла (расширение arx в имени файла и путь могут быть опущены); <сбой> — любое выражение, которое будет вычислено (выполнено) при невозможности выгрузки файла.

Возвращаемое значение — строка с именем файла, при успешной выгрузке. При сбое выгрузки возвращается результат вычисления выражения, заданного на случай ошибки в аргументе <сбой>.

Примеры:

- `(arxunload "geomcal.arx")` возвращает `"geomcal.arx"`
- `(arxunload "noarx" (setq na 22))` возвращает `22`, если файл noarx.arx не может быть выгружен (при этом в результате выполнения выражения, записанного в качестве второго аргумента, переменной `na` присваивается `22`)

1.17.13. Функция *autoload*

Задает имена команд, первое обращение к любой из которых требует загрузку файла приложения (с расширением lsp, fas или vlx).

```
(autoload <файл> <список>)
```

Аргументы: <файл> — текстовая строка, задающая имя файла приложения; <список> — список из текстовых строк с именами команд.

Возвращаемое значение — `nil`.

Как только пользователь в первый раз набирает имя команды, включенное в список, фигурирующий в качестве второго аргумента функции `autoload`, так AutoCAD выполняет загрузку указанного файла (аналогично функции `load`). В загружаемом файле должна быть обязательно описана функция с префиксом `c:` и соответствующая вызываемой команде, иначе может произойти сбой системы (возможно зацикливание).

Пример:

- `(autoload "rad.fas" '("RAC" "RBC"))` возвращает `nil`

Когда пользователь впервые введет команду RAC, то будет загружен файл rad.fas (загрузка заканчивается сообщением об инициализации) и затем станет выполняться команда, например (в русской версии):

```
Инициализация...
На слое HOLES нет кругов.
```

Здесь первое сообщение — сообщение об инициализации (загрузке), второе — уже сообщение исполняемой команды.

1.17.14. Функция *autoarxload*

Задает имена команд, любая из которых при первом обращении требует загрузку файла ARX-приложения.

```
(autoarxload <файл> <список>)
```

Аргументы: `<файл>` — текстовая строка, задающая имя файла приложения ObjectARX; `<список>` — список из текстовых строк с именами команд.

Возвращаемое значение — `nil`.

Данная функция аналогична функции `autoload`, но выполняет при необходимости загрузку файла с расширением arx (как это делает функция `arxload`).

Пример:

- `(autoarxload "rad.arx" '("RAC" "RBC"))` возвращает `nil`

1.17.15. Функция *vl-arx-import*

Позволяет VLX-приложению, имеющему собственное пространство имен (о сборке приложения с собственным именным пространством *см. разд. 2.4.3*), импортировать в свое пространство функцию (или функции) ObjectARX или ADSRX[7].

```
(vl-arx-import [<приложение>])
```

Аргумент `<приложение>` — символ имени функции (с апострофом впереди) или текстовая строка с именем файла, содержащего одно или несколько ARX-приложений.

Возвращаемое значение не определено.

Примеры:

- `(vl-arx-import 'c:pritent)`
- `(vl-arx-import "pritent.arx")`

1.17.16. Функция *vl-doc-export*

Экспортирует в пространство имен документа (рисунка) функцию из имеющего собственное именное пространство (об пространствах *см. разд. 1.17.32* и *2.4.3*) VLX-приложения, в котором данная функция описана.

```
(vl-doc-export '<функция>)
```

Без этой операции загружаемые в один документ VLX-приложения с отдельными именными пространствами не могут пользоваться функциями друг друга (см. функцию `vl-doc-import`), поскольку эти функции остаются локальными (доступными только внутри VLX-приложения).

Аргумент: символ с именем функции.

Возвращаемое значение не определено.

---

[7] Система разработки приложений на языке C, предшествовавшая ObjectARX.

Пример:

- `(vl-doc-export 'c:pritent)` дает возможность другому VLX-приложению с отдельным именным пространством импортировать к себе функцию `c:pritent`

Не следует использовать функцию `vl-doc-export` внутри функции `defun` и других аналогичных средств *(см. разд. 1.16.4—1.16.7)* создания пользовательских функций.

1.17.17. Функция *vl-doc-import*

Импортирует из пространства имен документа в собственное именное пространство текущего VLX-приложения заданные функции, ранее экспортированные другим VLX-приложением с отдельным именным пространством (см. функцию `vl-doc-export`).

```
(vl-doc-import <приложение> ['<функция1> ['<функция2>] ... ['<функцияN>]...]])
```

Аргументы: `<приложение>` — текстовая строка с именем приложения (указывается без пути и без расширения vlx), из которого импортируются функции; `<функция1>`, `<функция2>`, . . ., `<функцияN>` — символы с именами импортируемых функций.

Возвращаемое значение не определено.

Пример:

- `(vl-doc-import "makedesign" 'c:pritent 'trlow)` импортирует из VLX-приложения makedesign.vlx в другое VLX-приложение функции `c:pritent`, `trlow`

Не следует использовать функцию `vl-doc-import` внутри функции `defun` и других аналогичных средств *(см. разд. 1.16.4—1.16.7)* создания пользовательских функций.

1.17.18. Функция *vl-list-exported-functions*

Выводит список экспортированных функций.

```
(vl-list-exported-functions [<приложение>])
```

Аргумент — текстовая строка с именем приложения (путь и расширение vlx в строку не включаются).

Возвращаемое значение — список из текстовых строк с именами экспортированных функций приложения, имя которого задано в качестве значения аргумента `<приложение>`. Возвращается `nil`, если не обнаружены экспортированные функции. Если аргумент `<приложение>` не задан или задан равным `nil`, то возвращается список всех экспортированных (подгруженных) в текущий документ функций, кроме функций, экспортированных из VLX-приложений с отдельными именными пространствами.

Примеры:

- `(vl-list-exported-functions)` возвращает список всех экспортированных функций, в который входят функции, определяющие дополнительные команды AutoCAD, не входящие в основное ядро системы:

```
("C:-BLOCKREPLACE" "C:-BLOCKTOXREF" "C:-CDORDER" "C:-COPYTOLAYER"
"C:-LAYOUTMERGE" "C:-LMAN" "C:-LMANMODE" "C:-OVERKILL" "C:-REDIRMODE"
"C:-TCASE" "C:-XLIST" "C:3D" "C:3DARRAY" "C:ACADINFO" "C:ACETUCS-BACK"
"C:ACETUCS-BOTTOM" "C:ACETUCS-FRONT" "C:ACETUCS-LEFT" "C:ACETUCS-RIGHT"
"C:ACETUCS-TOP" "C:ADDVARS2SCR" "C:AI_BOX" "C:AI_CIRCTAN" "C:AI_CONE"
"C:AI_CUSTOM_SAFE" "C:AI_DIM_TEXTABOVE" "C:AI_DIM_TEXTCENTER"
"C:AI_DIM_TEXTHOME" "C:AI_DISH" "C:AI_DOME" "C:AI_FMS" "C:AI_MESH"
"C:AI_MOLC" "C:AI_PRODUCT_SUPPORT" "C:AI_PRODUCT_SUPPORT_SAFE"
"C:AI_PSPACE" "C:AI_PYRAMID" "C:AI_SELALL" "C:AI_SPHERE" "C:AI_TILEMODE1"
"C:AI_TORUS" "C:AI_TRAINING_SAFE" "C:AI_WEDGE" "C:ALIASEDIT" "C:ALIGNSPACE"
"C:ARCTEXT" "C:ATEXT" "C:ATTIN" "C:ATTOUT" "C:ATTREDEF" "C:BCOUNT"
"C:BEXTEND" "C:BLOCK?" "C:BLOCKREPLACE" "C:BLOCKTOXREF" "C:BREAKLINE"
"C:BSCALE" "C:BTRIM" "C:BURST" "C:CDORDER" "C:CHSPACE" "C:CHURLS"
"C:CLIPIT" "C:COPYM" "C:COPYTOLAYER" "C:DATE" "C:DDEMODES" "C:DDGRIPS"
"C:DDRMODES" "C:DDSELECT" "C:DIMEX" "C:DIMIM" "C:DIMREASSOC" "C:EDGE"
"C:EDITTIME" "C:ETBUG" "C:EXOFFSET" "C:EXPLAN" "C:EXPRESSMENU"
"C:EXPRESSTOOLS" "C:EXTRIM" "C:FASTSEL" "C:FLATTEN" "C:FS" "C:FSCREEN"
"C:FSMODE" "C:FULLSCREEN" "C:GATTE" "C:GETSEL" "C:GIFIN" "C:GOTOURL"
"C:IMAGEAPP" "C:IMAGEEDIT" "C:IMAGEOVERLAP" "C:LAYCUR" "C:LAYDEL"
"C:LAYFRZ" "C:LAYISO" "C:LAYLCK" "C:LAYMCH" "C:LAYMRG" "C:LAYOFF" "C:LAYON"
"C:LAYOUTMERGE" "C:LAYTHW" "C:LAYULK" "C:LAYUNISO" "C:LAYVPI" "C:LAYVPMODE"
"C:LAYWALK" "C:LMAN" "C:LMANMODE" "C:LSP" "C:LSPDUMP" "C:LSPSURF"
"C:MKLTYPE" "C:MKSHAPE" "C:MOCORO" "C:MOVEBAK" "C:MPEDIT" "C:MSTRETCH"
"C:MVSETUP" "C:NCOPY" "C:OVERKILL" "C:PCXIN" "C:PLT2DWG" "C:PROPULATE"
"C:PSBSCALE" "C:PSTSCALE" "C:QLATTACH" "C:QLATTACHSET" "C:QLDETACHSET"
"C:QQUIT" "C:REDIR" "C:REDIRMODE" "C:REPURLS" "C:REVERT" "C:RTEDIT"
"C:RTEXT" "C:RTEXTAPP" "C:RTUCS" "C:SAVEALL" "C:SHOWURLS" "C:SHP2BLK"
"C:SSX" "C:SUPERHATCH" "C:SYSVDLG" "C:TCASE" "C:TCIRCLE" "C:TCOUNT"
"C:TEXTFIT" "C:TEXTMASK" "C:TEXTUNMASK" "C:TFHELP" "C:TFRAMES" "C:TIFFIN"
"C:TJUST" "C:TORIENT" "C:TREX" "C:TSCALE" "C:TSPACEINVADERS" "C:TUTCLEAR"
"C:TUTDEMO" "C:TXT2MTXT" "C:TXTEXP" "C:VARS2SCR" "C:VLISP" "C:VPSCALE"
"C:VPSYNC" "C:XDATA" "C:XDLIST" "C:XLIST")
```

- `(vl-list-exported-functions "PRITENT")` возвращает `("C:PRITENT")`

Первый пример показывает, каким образом можно получить список всех дополнительно описанных команд, которые были подгружены к основному ядру системы AutoCAD (сюда входят, например, такие важные команды, как 3D (3M), VLISP, MVSETUP (ФОРМАТЛ), EXPRESSTOOLS, а также команды, созданные и загруженные в сеансе редактирования пользователем).

1.17.19. Функция *vl-list-loaded-vlx*

Выводит список всех VLX-приложений, имеющих в текущем документе собственные пространства имен.

```
(vl-list-loaded-vlx)
```

Возвращаемое значение — список из символов, являющихся именами функций. При отсутствии в текущем рисунке загруженных VLX-приложений с собственными именными пространствами возвращается `nil`.

Пример:

- `(vl-list-loaded-vlx)` может вернуть `(PRITENT)`, если в текущий рисунок было загружено одно имеющее отдельное пространство имен VLX-приложение pritent.vlx

1.17.20. Функция *vl-vlx-loaded-p*

Проверяет, загружено ли VLX-приложение, имеющее собственное пространство имен.

```
(vl-vlx-loaded-p <приложение>)
```

Аргумент — текстовая строка с именем приложения (путь и расширение vlx в строку не включаются).

Возвращаемое значение — `T`, если анализируемое приложение с отдельным именным пространством загружено, и `nil` — в других случаях.

Рассмотрим пример. Предположим, в текущий документ загружено приложение pritent.vlx. Тогда:

- `(vl-vlx-loaded-p "pritent")` возвратит `T`, если приложение pritent.vlx было собрано как приложение с собственным именным пространством, и `nil`, если pritent.vlx не имеет собственного именного пространства (в этом случае его функции и переменные загружаются в пространство имен рисунка).

1.17.21. Функция *vl-unload-vlx*

Выгружает VLX-приложение, имеющее собственное пространство имен.

```
(vl-unload-vlx <приложение>)
```

Аргумент — текстовая строка с именем приложения (путь и расширение vlx в строку не включаются).

Возвращаемое значение — `T`, если функции `vl-unload-vlx` удалось выгрузить указанное приложение с отдельным пространством имен. В случае сбоя выгрузки возвращаемое значение отсутствует, и выводится сообщение об ошибке.

Пример:

- `(vl-unload-vlx "PRITENT")` возвращает `T`

Если произошла ошибка (приложение не загружено или загружено, но не с собственным именным пространством), то выдается следующее сообщение:

; error: LISP Application is not found PRITENT

(; ошибка: LISP-приложение не найдено PRITENT)

1.17.22. Функция *vl-get-resource*

Выводит содержимое текстового файла, включенного в сборку VLX-приложения.

```
(vl-get-resource <файл>)
```

Аргумент — текстовая строка с именем текстового файла, включенного в приложение (расширение txt указывать не надо).

Возвращаемое значение — текстовая строка с содержимым TXT-файла, включенного в VLX-приложение.

Пример:

- `(vl-get-resource "girls")` может внутри работающего VLX-приложения вернуть строку (строки): `"Три девицы под окном\nПряли поздно вечерком."`

1.17.23. Функция *vl-vbaload*

Загружает DVB-приложение.

```
(vl-vbaload <файл>)
```

Аргумент — текстовая строка с именем DVB-приложения (приложения на языке Visual Basic for Applications).

Возвращаемое значение не определено.

Пример (файл drawline.dvb должен быть в вашей версии AutoCAD 2006 в папке Sample\Vba):

- `(vl-vbaload "d:\\AutoCAD 2006\\Sample\\Vba\\drawline.dvb")` возвращает `nil` (в данном случае это не является признаком ошибки)

1.17.24. Функция *vl-vbarun*

Выполняет макрос VBA, загруженный в текущий рисунок.

```
(vl-vbarun <макрос>)
```

Аргумент — текстовая строка с именем макроса, загруженного в документ.

Возвращаемое значение — текстовая строка с именем макроса.

Пример:

- `(vl-vbaload "d:\\AutoCAD 2006\\Sample\\Vba\\drawline.dvb")` возвращает `nil`
- `(vl-vbarun "drawline")` вызывает диалоговое окно **AutoCAD VBA Line Sample**, приведенное на рис. 1.14.

В этом диалоговом окне щелчок по кнопке **Draw Line** (Рисовать отрезок) выполняет рисование отрезка из точки (1,1,0) в точку (5,5,0), а щелчок по кнопке **Done**

(Сделано) завершает работу макроса и выводит "drawline" в качестве возвращаемого значения функции vl-vbarun.

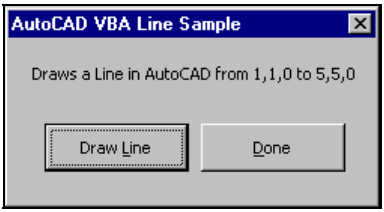


Рис. 1.14. Диалоговое окно **AutoCAD VBA Line Sample**

1.17.25. Функция *vl-bb-set*

Задает новое значение переменной из внедокументного пространства имен.

```
(vl-bb-set '<символ> <значение>)
```

Аргументы: <символ> — имя изменяемой переменной; <значение> — значение, присваиваемое переменной (не может быть именем функции).

Возвращаемое значение функции — значение второго аргумента.

Внедокументное пространство имен используется для передачи значений между документами (рисунками). Пример:

- (setq curnumber 19) возвращает 19 (значение переменной curnumber в именном пространстве текущего документа)
- (vl-bb-set 'bbnum curnumber) возвращает 19 (значение переменной bbnum во внедокументном именном пространстве)

См. также функцию vl-bb-ref.

1.17.26. Функция *vl-bb-ref*

Читает значение переменной из внедокументного пространства имен.

```
(vl-bb-ref '<символ>)
```

Аргумент — имя переменной (символ).

Возвращается значение прочитанной переменной. Если переменная с указанным именем во внедокументном пространстве не найдена, то возвращается nil.

Пример (продолжение примера из предыдущего раздела):

- (setq ournum (vl-bb-ref 'bbnum)) возвращает 19 (читает значение переменной bbnum из внедокументного именного пространства и присваивает его переменной ournum в текущем документе)

1.17.27. Функция *vl-doc-set*

Изменяет значение переменной в пространстве имен текущего документа (рисунка).

```
(vl-doc-set '<символ> <значение>)
```

Аргументы: `<символ>` — имя переменной (символа) из пространства имен текущего рисунка; `<значение>` — любое значение.

Возвращается значение второго аргумента.

Эта функция может применяться внутри VLX-приложений, имеющих собственное именное пространство, для изменения значений глобальных переменных текущего документа. Если функция `vl-doc-set` используется внутри VLX-приложения без собственного именного пространства, то тогда ее действие аналогично действию функции `set`.

Пример:

- `(setq curlist '(1 2 3))` возвращает `(1 2 3)`

  `(vl-doc-set 'doclist curlist)` возвращает `(1 2 3)`

См. также функцию `vl-doc-ref`.

1.17.28. Функция *vl-doc-ref*

Читает значение переменной из пространства имен текущего документа (рисунка).

```
(vl-doc-ref '<символ>l)
```

Аргумент `<символ>` — имя переменной (символа) из пространства имен текущего рисунка.

Возвращается значение прочитанной переменной из пространства имен текущего рисунка.

Эта функция позволяет VLX-приложениям со своим именным пространством импортировать значения переменных из пространства имен текущего рисунка.

Пример (продолжение примера из предыдущего раздела):

- `(setq list2 (vl-doc-ref 'doclist))` возвращает `(1 2 3)`

Функция `vl-doc-ref` является обратной по действию для функции `vl-doc-set`.

1.17.29. Функция *vl-load-all*

Данная функция уже была упомянута в *разд. 1.2.5* в свете загрузки LSP-файлов. Вернемся к ней еще раз для более подробного рассмотрения.

```
(vl-load-all <файл>)
```

Загружает LISP-файл (не обязательно с расширением lsp) во все уже открытые документы и обеспечивает его автоматическую загрузку во все рисунки, которые будут открыты в данном сеансе редактирования в системе AutoCAD.

Аргумент — строка с именем загружаемого файла. Расширение файла опускать нельзя. Если полный путь к файлу в аргументе не указан, то AutoCAD пытается найти его сначала в текущем каталоге, а затем в папках, указанных в путях поиска файлов поддержки, указанных в настройке системы AutoCAD.

Возвращаемое значение не определено. В случае ошибки выводится сообщение о сбое загрузки.

Пример:

- `(vl-load-all "nofile.vlx")` вызывает в случае сбоя сообщение:

  ; error: LOAD failed: "nofile.vlx"

  (; ошибка: сбой при выполнении LOAD: "nofile.vlx")

См. также использование файла acaddoc.lsp в *разд. 1.19*.

1.17.30. Функция *vl-propagate*

Копирует значение переменной текущего документа во все другие открытые рисунки и обеспечивает копирование переменной во все рисунки, которые еще будут открыты в данном сеансе редактирования в системе AutoCAD.

```
(vl-propagate '<символ>)
```

Аргумент `<символ>` — имя переменной (символа) из пространства имен текущего рисунка.

Возвращаемое значение не определено.

Пример:

- `(vl-propagate 'doc1list)` копирует значение глобальной переменной `doc1list` из текущего рисунка в глобальные переменные `doc1list` всех других рисунков (уже открытых или которые будут открыты в данном сеансе редактирования)

Другой способ передачи значений переменных — использование функций `vl-bb-set` и `vl-bb-ref`, работающих с внедокументным именным пространством.

1.17.31. Глобальные и локальные переменные

Рассмотрим в данном разделе работу в одном документе (рисунке). Переменные, используемые в приложениях, загружаемых в данный документ, делятся на глобальные и локальные.

К *локальным переменным* относятся переменные, имена которых заданы при описании функций либо в перечне аргументов, либо в перечне рабочих перемен-

ных *(см. разд. 1.16.2, 1.16.4)*. Такие переменные существуют только внутри функции, в которой они определены, и их значения не передаются наружу.

Рассмотрим определение функции `fkos`, приведенное в листинге 1.29.

Листинг 1.29. Пример функции, использующей локальные переменные

```
; Глава 01\Листинг 1.29.lsp
(defun fkos (r1 pp1 / pp2 vo)
  (command "_.CIRCLE" pp1 r1)
  (setq vo (entlast))
  (initget 1)
  (setq pp2 (getpoint "\nЗадайте точку: "))
  (command "_.COPY" vo "" pp1 pp2)
  (princ "\nОкружности построены. ")
  (princ); тихий выход
); конец defun
```

В этой функции определены четыре локальные переменные: `r1`, `pp1`, `pp2` и `vo`. Первые две из них являются аргументами функции, а две последние — рабочими переменными.

Функция `fkos` работает следующим образом. Сначала она строит окружность в точке, указанной в качестве значения аргумента `pp1`, и радиусом, заданным в качестве значения аргумента `r1`. Примитив построенной окружности запоминается в рабочей переменной `vo`.

Затем запрашивается ввод точки (пустое значение в качестве ответа на запрос не допускается) и введенная точка (точнее, список с ее координатами) запоминается в переменной `pp2`, которая является рабочей. После этого система AutoCAD копирует первую окружность таким образом, что центр новой окружности оказывается в точке `pp2`.

Попробуйте загрузить функцию `fkos` в новый рисунок, когда в *пространстве имен документа* (пространстве функций и переменных) еще нет ваших пользовательских символов, и выполнить ее — например, следующим образом:

```
(fkos 55.0 '(0.0 0.0))
```

В результате работы функции `fkos` в рисунке будут построены две окружности (одна — с центром в точке (0,0), а вторая — с центром в той точке, которую вы укажете в ответ на запрос `"Задайте точку: "`). Радиус обеих окружностей будет равен 55.

Если попробуем в командной строке AutoCAD прочитать значения переменных с помощью операции `!` (восклицательный знак), то выясним, что значения всех локальных переменных (`r1`, `pp1`, `pp2` и `vo`) равны `nil` (операции `!r1`, `!pp1`, `!pp2`, `!vo` возвращают `nil`) — иначе говоря, все четыре переменные не определены. Связано это с тем, что значения локальных переменных вне функции `fkos` вам не дос-

тупны. Таким образом, указанные переменные не занимают память в пространстве имен рисунка, которое хранит значения всех символов (переменных и функций).

Остальные переменные, которые не являются локальными, считаются *глобальными*, и им отводится специальное место в именном пространстве документа (рисунка). Отсюда вы можете читать их значения с помощью операции `!`. Сейчас в именном пространстве рисунка есть уже один символ — `fkos`. Если вы прочитаете его значение, то получите примерно следующее: `#<SUBR @0acc1a50 FKOS>`.

Создадим в рисунке две глобальные переменные, введя с клавиатуры следующее выражение:

```
(setq mainradius 17.89 mainpoint (list 60.345 12.107))
```

Поскольку указанное здесь выражение является изолированным, то оно не входит в тело никакой функции, и переменные `mainradius` и `mainpoint` являются глобальными. Их значения можно прочитать:

- `!mainradius` возвращает `17.89`
- `!mainpoint` возвращает `(60.345 12.107)`

Эти две переменные занимают свое место в пространстве имен документа и могут в любое время использоваться пользователем или приложениями, не имеющими собственного пространства имен. Например, пользователь может ввести:

```
(fkos (+ mainradius 1.1) mainpoint)
```

Это выражение выполнит функцию `fkos` и построит в текущем рисунке две окружности с радиусом 18.99.

Изменим пример, приведенный в листинге 1.29, следующим образом (листинг 1.30).

Листинг 1.30. Пример функции, использующей локальные и глобальные переменные

```
; Глава 01\Листинг 1.30.lsp
(defun fkos1 (r1 pp1 / pp2)
  (command "_.CIRCLE" pp1 r1)
  (setq vo (entlast))
  (initget 1)
  (setq pp2 (getpoint "\nЗадайте точку: "))
  (command "_.COPY" vo "" pp1 pp2)
  (princ "\Окружности построены. ")
  (princ); тихий выход
); конец defun
```

Функция `fkos1` отличается от функции `fkos` тем, что переменная `vo` перестала быть локальной. В таком случае она автоматически становится глобальной. Если

загрузить и выполнить функцию `fkos1`, то после выполнения можно будет прочитать значение переменной `vo`. Оно уже не будет равно `nil`:

`!vo` возвратит `<Entity name: 7ef80008>`

Таким образом, появилась глобальная переменная `vo`, расположенная в памяти рисунка (в его пространстве имен). Как ее применить, зависит уже от пользователя, работающего с данным документом.

Глобальные переменные удобны для использования в приложениях. Однако избыточное количество глобальных переменных отнимают место в памяти рисунка. Освободить участок памяти, занимаемый в рисунке глобальной переменной, можно, присвоив этой переменной значение `nil`. При многократном освобождении памяти может быть полезна операция сборки мусора (см. функцию `gc`), которая дает возможность использовать эти участки другим программам.

При одновременной загрузке приложений, созданных разными разработчиками, могут возникать проблемы совместимости, если одни и те же имена глобальных переменных одновременно фигурируют в разных функциях с разным назначением. Поэтому разным разработчикам можно рекомендовать присваивать имена глобальным переменным и функциям с разными префиксами.

1.17.32. Пространства имен

Из предыдущего раздела видно, что в каждом документе (рисунке) есть область памяти (пространство имен), в котором располагаются символы документа, в число которых входят глобальные переменные и функции. Ими могут пользоваться все приложения, загружаемые в данный рисунок, кроме тех VLX-приложений, которые имеют собственное именное пространство. Как собрать приложение, имеющее собственное именное пространство, сказано в *разд. 2.4.3*.

VLX-приложения с собственными именными пространствами уже не имеют проблемы совместимости, т. к. глобальные переменные одного приложения недоступны другому без специальных действий.

Однако для того, чтобы получить доступ к символам документа, VLX-приложение должно импортировать к себе (в свое пространство имен) необходимые функции (см. функцию `vl-doc-import`) и значения переменных (см. функцию `vl-doc-ref`).

Если VLX-приложению с собственным именным пространством нужно передать свои символы в пространство имен документа, то в приложении должны быть использованы функции `vl-doc-export` (экспортирует функции приложения в пространство имен рисунка) и `vl-doc-set` (экспортирует значения глобальных переменных).

VLX-приложения, собранные без собственного именного пространства, все свои символы (функции и глобальные переменные) размещают в пространстве имен документа, к которому всегда имеют доступ.

1.17.33. Проблемы многодокументного режима

Существующая версия языка AutoLISP поддерживает многодокументный режим с ограничением, которое вызвано тем, что LISP-приложение загружается и выполняется в активном документе. Если активизировать другой рисунок, то выполнение программы, запущенной в предыдущем рисунке, приостановится. С помощью средств ActiveX можно работать и с другими рисунками, но не активизируя их.

Если ваше LISP-приложение, выполняемое в одном рисунке, захочет открыть другой рисунок с помощью команды OPEN (ОТКРЫТЬ), то ничего не получится:

`(command "_.OPEN")` вернет `nil`,

а попытка передать команде еще и имя файла:

`(command "_.OPEN" "Draw24")` вызовет сообщение о неизвестной команде Draw24.

Во многих случаях может выручить передача управления в среду VBA, в которой поддерживаются операции многодокументного режима. Например:

`(command "_.VBASTMT" "AcadApplication.Documents.Open \"Draw24.dwg\"")` успешно откроет рисунок Draw24, если он на самом деле существует. В данном примере команда VBASTMT позволяет выполнить выражение на языке Visual Basic for Applications. Аналогичную функцию выполняет метод `Eval`, рассматриваемый в *главе 7. См. также метод* `Open` *семейства* `Documents` *(глава 7)*.

Многодокументный режим в системе AutoCAD разрешается или не разрешается с помощью системной переменной SDI. Если переменная SDI имеет значение 1, то многодокументный режим запрещен. При таком значении системной переменной SDI попытка открыть следующий рисунок вызовет операцию закрытия предыдущего открытого рисунка.

Для того чтобы получить возможность в одном сеансе редактирования открывать одновременно в разных окнах несколько документов (рисунков), нужно установить для системной переменной SDI значение 0 (это значение является значением, действующим по умолчанию).

Работая в многодокументной среде AutoLISP, пользователь при переходе из документа в документ должен сам обеспечить наличие в пространстве имен следующего документа необходимых функций и переменных. Для этого в каждом документе должны отдельно загружаться функции и формироваться значения переменных.

Для облегчения процесса загрузки LISP-приложения (т. е. файла с расширением lsp, fas или vlx) во все документы можно применять функцию `vl-load-all`.

Если необходимо передать копию значения какой-то переменной во все уже открытые рисунки, следует воспользоваться функцией `vl-propagate`. Она же передаст значения указанных глобальных переменных в те документы (рисунки), ко-

торые будут в дальнейшем открыты в том же сеансе редактирования, в котором была использована функция `vl-propagate`.

Внедокументное именное пространство (blackboard) играет роль своеобразного буфера и облегчает разрешение проблемы передачи значений глобальных переменных. Если из какого-то рисунка с помощью функции `vl-bb-set` во внедокументное пространство имен передано значение какой-то переменной, то его можно будет прочитать из любого другого рисунка с помощью функции `vl-bb-ref`.

1.17.34. Функция *startapp*

Выполняет в среде AutoCAD другое приложение Windows. Синтаксис функции следующий:

```
(startapp <приложение> [<параметр>])
```

Оба аргумента функции являются строками. Функция запускает приложение и, если второй аргумент был задан, передает ему строку с параметрами.

Возвращаемое значение — целое число (положительное при успешном запуске приложения и `0` — при неудачном).

Пример:

- `(startapp "Notepad")` — открывает в программе Блокнот новый текстовый файл (по выходе возвращает `33`)

1.18. Другие функции

В данном разделе рассматриваются еще несколько функций, имеющих практическую ценность при разработке приложений пользователя. Кроме того, в *главах 3—9* будет рассмотрено большое количество функций для работы с такими объектами, как меню, диалоговые окна, справочная система, а также функции, реализующие технологию ActiveX.

1.18.1. Функция *osnap*

Изменяет указанную точку в соответствии с режимом объектной привязки.

```
(osnap <точка> <режим>)
```

Аргументы: `<точка>` — список с координатами двумерной или трехмерной точки в текущей системе координат; `<режим>` — строка с перечисляемыми через запятую (без пробелов) именами одновременно действующих режимов привязки (`"_end"`, `"_mid"`, `"_int"`, `"_nea"`, `"_nod"` и т. д.).

Возвращаемое значение — список из трех вещественных чисел (координаты трехмерной точки в текущей системе координат, уточненной с учетом заданных

режимов объектной привязки). Если точка определена неудачно и устройство указания в заданных условиях ничего не выбрало, то возвращается `nil`.

На результат операции оказывают влияние текущий вид, видимые около указанной точки объекты рисунка и значение системной переменной APERTURE, устанавливающей размер прицела устройства указания.

Пример:

- `(osnap '(16.32 4.782) "_end,_mid")` возвращает координаты трехмерной точки, уточненной с помощью применяемых одновременно функций объектной привязки к конечной и средней точкам

1.18.2. Функция *redraw*

Перерисовывает указанный примитив в текущем видовом экране или все объекты, видимые в этом видовом экране.

```
(redraw [<примитив> [<режим>]])
```

Аргументы: `<примитив>` — имя примитива, `<режим>` — целое число (номер режима для перерисовки объекта — табл. 1.9).

Возвращаемое значение — `nil`.

Если значение аргумента `<режим>` не указано, то выполняется перерисовка объекта в видовом экране, аналогично действию команды REDRAW (ОСВЕЖИТЬ). Если опущены оба аргумента, то перерисовываются все объекты, видимые в текущем видовом экране.

Таблица 1.9. *Коды режимов функции* `redraw`

Код	Назначение
1	Показать примитив
2	Спрятать примитив (перерисовать цветом фона)
3	Подсветить примитив (выделить пунктиром)
4	Снять подсветку примитива

Коды режимов можно использовать и с отрицательными значениями, действие которых аналогично действию соответствующих положительных значений, но в случае если в качестве первого аргумента указан сложный примитив (полилиния или вхождение блока с атрибутами), то действие функции не распространяется на подпримитивы (при положительных значениях — распространяется).

Примеры:

- `(redraw e1 2)` делает примитив `e1` невидимым
- `(redraw e1 1)` восстанавливает видимость примитива `e1`

- `(redraw e1 3)` подсвечивает примитив `e1`
- `(redraw e1 4)` убирает подсветку примитива `e1`

Если ваша программа подсвечивает какие-то примитивы, то необходимо обеспечить в программе и снятие их подсветки.

Команда REGEN (РЕГЕН) аннулирует действие функции `redraw`, восстанавливая обычную видимость примитивов.

1.18.3. Функция *graphscr*

Активизирует графический экран системы AutoCAD (снимает фокус с текстового окна, если оно было активизировано).

```
(graphscr)
```

Возвращаемое значение — `nil`.

Действие функции аналогично действию команды GRAPHSCR (ГРАФЭКР).

1.18.4. Функция *textscr*

Активизирует текстовое окно текущего документа системы AutoCAD (открывает это окно, если оно было закрыто).

```
(textscr)
```

Возвращаемое значение — `nil`.

Действие функции аналогично действию команды TEXTSCR (ТЕКСТЭКР).

1.18.5. Функция *textpage*

Активизирует текстовое окно.

```
(textpage)
```

Возвращаемое значение — `nil`.

Действие функции аналогично действию функции `textscr` и команды TEXTSCR (ТЕКСТЭКР).

1.18.6. Функция *grtext*

Записывает временный текст в строку состояния или бокс экранного меню.

```
(grtext [<бокс> <текст> [<подсветка>]])
```

Аргументы: `<бокс>` — целое число, задающее номер бокса или строку режимов (см. далее), куда будет выведен текст; `<текст>` — текстовая строка; `<подсветка>` — целое число (положительное число — признак подсветки бокса экранного меню, как будто он выбран с помощью устройства позиционирования).

Возвращается значение второго аргумента. В случае ошибки или отсутствия аргументов возвращается `nil`.

Вызов функции `grtext` без аргументов удаляет временные тексты, выведенные в строку режимов и боксы экранного меню.

Отрицательные значения аргумента `<бокс>` (–1 или –2) указывают на запись текста в строку состояния, положительные или 0 — на запись в боксы экранного меню.

При значении аргумента `<бокс>` –1 временный текст выводится в начало строки состояния, при этом предыдущее содержимое строки сдвигается вправо.

Замечание

Для вывода постоянного текста в строку состояния следует пользоваться системной переменной MODEMACRO.

Вывод в зону счетчика координат строки состояния осуществляется при значении аргумента `<бокс>` –2 и может быть выполнен либо в режиме активизированного текстового окна AutoCAD, либо при положении устройства указания в неактивном видовом экране (иначе постоянное считывание счетчика координат сбросит введенный текст). Если счетчик координат отключен (например, с помощью функциональной клавиши <F6>), то текст, выведенный с помощью функции `grtext` в зону счетчика координат, тоже будет сохраняться, хотя и в приглушенном виде.

Системная переменная SCREENBOXES указывает максимальное количество боксов действующего экранного меню, которое зависит от настройки системы AutoCAD и настройки экрана монитора. Номер 0 соответствует самому верхнему боксу экранного меню. Номера других боксов при движении сверху вниз получаются добавлением 1. Поэтому допустимыми значениями аргумента `<бокс>` функции `grtext` при работе с экранным меню являются целые числа от 0 до max–1, где max — это значение системной переменной SCREENBOXES (обычно 27).

Неверные значения аргумента `<бокс>` (больше максимального или меньше –2) игнорируются, и функция возвращает `nil`.

При положительном значении аргумента `<подсветка>` значение аргумента `<текст>` игнорируется, причем бокс с указанным номером не меняет своего текста, а подсвечивается, как будто он выбран с помощью устройства указания. При нулевом значении аргумента `<подсветка>` значение аргумента `<текст>` тоже игнорируется, но с указанного бокса снимается подсветка.

При отрицательном значении аргумент `<подсветка>` игнорируется (работает аргумент `<текст>`; если у бокса была подсветка, то она снимается). Если указатель мыши находится в зоне экранного меню, то аргумент `<подсветка>` также игнорируется.

В листинге 1.31 приведена функция `show`, иллюстрирующая различные варианты вывода временного текста с помощью функции `grtext`.

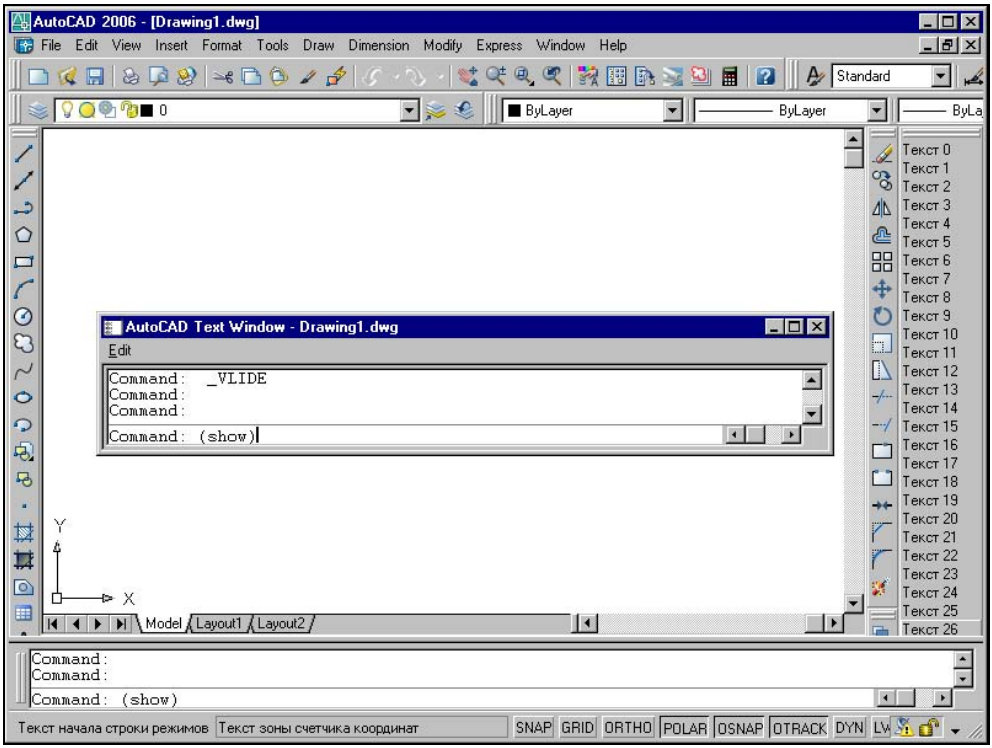


Рис. 1.15. Вывод текстовых строк с помощью функции `grtext`

Листинг 1.31. Примеры использования функции `grtext`

```
; Глава 01\Листинг 1.31.lsp
(defun show ( / i nb)
; Вывод в строку состояния
(grtext -1 "Текст начала строки режимов")
(grtext -2 "Текст зоны счетчика координат")
(setq i -1 nb (getvar "SCREENBOXES"))
; Вывод в экранное меню
(if (> nb 0)
  (progn
    (repeat nb
      (setq i (1+ i))
      (grtext i (strcat "Текст " (itoa i)))
    )
; Игра с подсветкой бокса 26
    (repeat 500000
```

```
; Подсвечиваем бокс 26
      (grtext 26 "+++" 1)
    )
; Снимаем подсветку с бокса 26
    (grtext 26 "---" 0)
  ); progn
); if
(textscr); оставляем текстовое окно
(princ); тихий выход
); defun
```

На рис. 1.15 показан результат выполнения функции `show`, выводящей текст в строку состояния и заполняющей по очереди все боксы экранного меню. Последний видимый на экране бокс с номером 26 подсвечен (выделен). Для того чтобы подсветка быстро не исчезла, бокс подсвечивается 500 000 раз.

1.18.7. Функция *grdraw*

Рисует в текущем видовом экране временный отрезок между двумя точками. Нарисованный отрезок не является примитивом. Может быть удален с помощью функции `redraw` или команд REDRAW (ОСВЕЖИТЬ) и REGEN (РЕГЕН).

```
(grdraw <из> <в> <цвет> [<подсветка>])
```

Аргументы: `<из>` и `<в>` — списки из двух или трех вещественных чисел (координаты двумерных или трехмерных точек в текущей системе координат); `<цвет>` — целое число, задающее номер цвета (значение –1 используется как обратный цвет при наложении рисуемого отрезка на существующие линии рисунка); `<подсветка>` — целое число, управляющее подсветкой рисуемого отрезка (0 — без подсветки, другие числа — с подсветкой пунктирной линией). Если значение аргумента `<подсветка>` опущено, то оно принимается равным 0.

Возвращаемое значение — `nil`.

Если концы заданного отрезка выходят за границы видового экрана, то отрезок рисуется только в видимой части экрана.

Пример:

- `(grdraw '(35.7 41.2) '(539.4 454.1) 5 1)` рисует пунктиром и синим цветом временный отрезок, не являющийся примитивом рисунка, и возвращает `nil`

1.18.8. Функция *grvecs*

Рисует группу временных отрезков в текущем видовом экране. Является обобщением функции `grdraw`.

```
(grvecs <отрезки> [<матрица>])
```

Аргументы: `<отрезки>` — список цветов и пар точек, определяющий координаты концов рисуемых отрезков в текущей системе координат и цвета отрезков (см. ниже); `<матрица>` — список, элементами которого являются четыре списка по четыре вещественных числа (матрица преобразования, используемая в функции `nentselp`).

Возвращаемое значение — `nil`.

Список цветов и пар точек имеет следующий вид:

```
([<цвет1>] <из1> <в1> [[<цвет2>] <из2> <в2> ... [[<цветN>] <изN> <вN>]...]])
```

Здесь `<цвет1>`, `<цвет2>`, …, `<цветN>` — целые числа с номерами цветов отрезков. Если к номеру цвета добавить знак минус, то отрезок будет нарисован с подсвечиванием. Когда номер цвета опущен, то действует цвет, установленный для предыдущего отрезка. Если номер цвета больше 255, тогда используется цвет, дополнительный к цвету примитивов рисунка, оказавшихся под отрезком.

`<из1>`, `<в1>`, `<из2>`, `<в2>`, … — списки с координатами двумерных или трехмерных точек, определяющие концы рисуемых отрезков.

Аргумент `<матрица>` задает дополнительное преобразование координат выводимых отрезков (см. матрицу преобразования, описанную для функции `nentselp`).

Пример:

- `(grvecs '(11 (0.0 0.0)(100.0 50.0) 12 (0.0 0.0)(100.0 100.0) 13 (0.0 0.0)(100.0 150.0)) '((1.0 0.0 0.0 20.0) (0.0 1.0 0.0 20.0) (0.0 0.0 1.0 0.0) (0.0 0.0 0.0 1.0)))` рисует из одной точки три отрезка, сдвигая их при этом с помощью матрицы преобразования на 20 мм по оси *X* и на 20 мм по оси *Y* и используя цвета с номерами 11, 12 и 13; возвращает `nil`

1.18.9. Функция *grclear*

Очищает текущий видовой экран (устаревшая функция, в вашей версии может не работать).

```
(grclear)
```

Возвращаемое значение — `nil`.

1.18.10. Функция *grread*

Вводит данные пользователя с любого устройства (клавиатура, мышь и др.). Требуется очень редко, поскольку функции с префиксом `get` *(см. разд. 1.10)* обеспечивают основные варианты пользовательского ввода.

```
(grread [<мышь>] [<флаг> [<курсор>]])
```

Аргументы: `<мышь>` — признак (если задан и не `nil`) возврата координат устройства позиционирования при его движении; `<флаг>` — флаг (целое число), являю-

щийся суммой значений установленных битовых флагов (см. ниже); <курсор> — целое число, определяющее тип курсора (см. ниже).

Возвращаемое значение — список из двух элементов. Первый элемент этого списка — код устройства, использованного для ввода; второй — либо целое число, либо список с координатами точки. См. ниже.

Значение аргумента <флаг> формируется из битовых флагов, перечисленных в табл. 1.10, которые установлены (включены) в данном обращении к функции grread.

Таблица 1.10. *Битовые флаги функции grread*

Номер бита	Значение битового флага	Назначение битового флага, если он включен
0	1	Возвращает координаты при движении устройства указания. Результат — список, первый элемент которого — число 5, второй — список с координатами (*X*, *Y*) считанной точки
1	2	Обрабатывает результаты ввода данных с любого устройства, в том числе нажатия функциональных и обычных клавиш. Не перемещает перекрестие, если пользователь нажмет кнопку устройства указания
2	4	Использует форму курсора, заданную аргументом <курсор>
3	8	Не выводит сообщение об ошибке, если пользователь нажал клавишу <Esc>

Значение аргумента <флаг> получается как сумма тех битовых флагов, которые включены, например: 1 = 1+0+0+0, 3 = 1+2+0+0, 7 = 1+2+4+0 и т. д.

В табл. 1.11 приведены возможные значения аргумента <курсор>, используемые, когда в аргументе <флаг> включен битовый флаг, соответствующий биту 2.

Таблица 1.11. *Значения аргумента <курсор> функции grread*

Значение	Описание
0	Показывает обычное перекрестие
1	Не показывает никакого курсора устройства указания
2	Показывает курсор в форме прицела (квадрата)

В табл. 1.12 приведены варианты возвращаемых значений, являющихся списками из двух элементов.

Таблица 1.12. *Элементы списка, возвращаемого функцией* `grread`

Первый элемент		Второй элемент	
Значение	**Описание типа ввода**	**Значение**	**Описание**
2	Ввод с клавиатуры	Любое	Буквенно-цифровой символ
3	Точка, переданная устройством указания	Трехмерная точка	Список с координатами точки
4	Пункт экранного или падающего меню, выбранный устройством указания	0—999 1000—1999 2000—2999, … 16000—16999	Номер бокса экранного меню Номер пункта падающего меню 1 Номер пункта 2 … Номер пункта падающего меню 16
5	Точка, переданная устройством указания (при включенном счетчике координат)	Трехмерная точка	Список с координатами
6	Выбранный пункт кнопочного меню	0—999 1000—1999 2000—2999 3000—3999	Номер пункта меню BUTTONS1 Номер пункта меню BUTTONS2 Номер пункта меню BUTTONS3 Номер пункта меню BUTTONS4
7	Выбранный пункт планшетного меню TABLET1	0—32767	Номер элемента меню
8	Выбранный пункт планшетного меню TABLET2	0—32767	Номер элемента меню
9	Выбранный пункт планшетного меню TABLET3	0—32767	Номер элемента меню
10	Выбранный пункт планшетного меню TABLET4	0—32767	Номер элемента меню
11	Выбранный пункт дополнительного меню AUX	0—999 1000—1999 2000—2999 3000—3999	Номер кнопки меню AUX1 Номер кнопки меню AUX2 Номер кнопки меню AUX3 Номер кнопки меню AUX4

***Таблица 1.12** (окончание)*

Первый элемент		Второй элемент	
Значение	**Описание типа ввода**	**Значение**	**Описание**
12	Нажатая кнопка устройства указания (следует за вводом типа 6 или типа 11)	Трехмерная точка	Список с координатами точки

Несколько дополнительных соображений по поводу работы с функцией `grread`.

Нажатие клавиши <Esc> при работающей функции `grread` прервет действие всех функций языка LISP (в том числе и самой `grread`), если не включен битовый флаг 8 (см. табл. 1.10).

Возвращаемые устройствами ввода данные помещаются в специальный буфер. Если пользователь нажмет кнопку мыши при положении устройства указания в зоне экранного или падающего меню, то функция `grread` вернет некоторый список. При следующем вызове функции `grread` первым элементом возвращаемого значения не сможет стать число 12.

Другая ситуация с выбором пункта кнопочного или дополнительного меню. Вслед за начальным возвращаемым значением (с первым элементом 6 или 11) в буфер возврата последует значение с первым элементом 12, которое в программе пользователя должно быть обязательно прочитано еще одним вызовом функции `grread`, например:

```
(setq return2 (grread (setq return1 (grread))))
```

Оба возвращенных значения, сохраненные в переменных `return1` и `return2`, далее могут быть проанализированы.

Пример:

- `(grread)` ждет ввода и возвращает, например, `(2 119)` в случае нажатия клавиши <W> или, например, `(3 (683.472 522.901 0.0))` в случае указания точки мышью в графическом экране
- `(grread T)` возвращает, например, список `(5 (1048.16 423.301 0.0))` с текущими координатами курсора

Интересным примером использования функции `gread` является программа стирания объектов под курсором при движении курсора по экрану, без нажатия кнопок *(см. главу 25 книги "САПР на базе AutoCAD — как это делается").*

1.18.11. Функция *initdia*

Заставляет некоторые команды системы AutoCAD, вызываемые из LISP-программы с помощью функций `command` и `vl-cmdf`, работать с диалоговым окном.

```
(initdia [<флаг>])
```

Аргумент <флаг> — целое число. Если значение аргумента опущено или задано не равным 0, то вызываемая из программы ближайшая (и только она) команда системы AutoCAD будет работать с диалоговым окном, если такая возможность в команде предусмотрена. Если функция initdia вызвана со значением аргумента, равным 0, то досрочно сбрасывается настройка на вызов диалогового окна в очередной команде.

Возвращаемое значение — nil.

Функция initdia оказывает влияние на программный вызов следующих команд: ATTDEF (АТОПР), ATTEXT (АТЭКСП), BHATCH (КШТРИХ), BLOCK (БЛОК), COLOR (ЦВЕТ), IMAGE (ИЗОБ), IMAGEADJUST (ИЗОБРЕГУЛ), INSERT (ВСТАВИТЬ), LAYER (СЛОЙ), LINETYPE (ТИПЛИН), MTEXT (МТЕКСТ), PLOT (ПЕЧАТЬ), RENAME (НОВОЕИМЯ), STYLE (СТИЛЬ), TOOLBAR (ПАНЕЛЬ), VIEW (ВИД).

Например:

(command "_.COLOR") выдает в командной строке запрос (без диалогового окна):

Enter default object color [Truecolor/COlorbook]:

(Цвет объектов по умолчанию [изПалитры/Альбом]:)

Однако если в программе перед выполнением команды COLOR (ЦВЕТ) ввести обращение к функции initdia, то ситуация изменится:

(initdia)(command "_.COLOR") уже вызывает диалоговое окно **Select Color** (Выбор цвета).

Функция initdia — одноразового действия. При следующем программном выполнении команды COLOR (ЦВЕТ) без вызова функции initdia она (команда) будет снова работать в режиме командной строки (без диалогового окна).

1.18.12. Функция *tablet*

Читает или изменяет текущую настройку (калибровку) планшета.

(tablet <код> [<ряд1> <ряд2> <ряд3> <нормаль>])

Аргументы: <код> — целое число (код операции, который может быть только 0 или 1); <ряд1>, <ряд2> и <ряд3> — три трехмерные точки (списки из трех чисел), определяющие матрицу преобразования точек, координаты которых считаны дигитайзером; <нормаль> — трехмерная точка (список с координатами в МСК), определяющая направление нормали к поверхности планшета.

Возвращаемое значение — список из пяти элементов с настройками планшета. Если калибровка невозможна или из-за ошибки в данных, или из-за того, что дигитайзер не установлен, или из-за отключения планшета с помощью системной переменной TABMODE — возвращается nil.

Код операции 0 означает чтение параметров настройки, а код 1 — их изменение. При коде операции 0 остальные параметры должны отсутствовать.

Третья координата аргумента `<ряд3>` должна быть равна 1, иначе AutoCAD сам изменит ее на 1.

Если в качестве значения аргумента `<нормаль>` задана ненормализованная точка, то AutoCAD ее сам нормализует (приводит к единичному вектору).

Примеры:

- `(tablet 0)` может вернуть `(1 (0.00561717 -0.000978942 -7.5171) (0.000978942 0.00561717 -9.17308) (0.0 0.0 1.0) (0.0 0.0 1.0))`
- `(tablet 1 '(1 0 0) '(0 1 0) '(0 0 1) '(0 0 1))` задает такую настройку планшета, при которой дигитайзер будет возвращать считанные координаты точек без дополнительного преобразования

1.18.13. Функция *acad-push-dbmod*

Запоминает в стеке текущее значение системной переменной DBMOD, отражающей состояние редактирования рисунка.

```
(acad-push-dbmod)
```

Возвращаемое значение — `T`.

Пример:

- `(acad-push-dbmod)` возвращает `T`

1.18.14. Функция *acad-pop-dbmod*

Восстанавливает из стека значение системной переменной DBMOD.

```
(acad-pop-dbmod)
```

Возвращаемое значение — `T` при успешном завершении и `nil` — при ошибочном.

Пример:

- `(acad-pop-dbmod)` возвращает `T`, если ранее была вызвана функция `acad-push-dbmod`

1.18.15. Функция *acdimenableupdate*

Управляет режимом автоматического обновления ассоциативных размеров.

```
(acdimenableupdate [<признак>])
```

Аргумент: `<признак>` — любое значение (важно только `nil` или нет). Значение по умолчанию — `T`.

Возвращаемое значение — предыдущее состояние режима автоматического обновления (`nil`, если был выключен, и `T`, если был включен).

Если аргумент <признак> равен nil, то устанавливается режим, при котором ассоциативный размер автоматически не обновляется после редактирования родительского примитива — для обновления необходимо выполнить команду DIMREGEN (РЗМРЕГЕН). Если <признак> имеет значение, отличное от nil, то устанавливается режим автоматического обновления.

Если аргумент опущен, то возвращается текущее состояние режима автоматического обновления ассоциативных размеров.

Примеры:

- (acdimenableupdate nil) отменяет автоматическое обновление ассоциативных размеров и возвращает T, если обновление перед этим действовало
- (acdimenableupdate) возвращает nil, если в текущий момент автоматическое обновление отключено
- (acdimenableupdate T) восстанавливает автоматическое обновление ассоциативных размеров и возвращает nil, если перед этим обновление было отключено

1.18.16. Функция *acet-layerp-mode*

Изменяет режим работы команды LAYERP (СЛОЙП). Действие функции аналогично работе команды LAYERPMODE (СЛОЙПРЕЖИМ).

```
(acet-layerp-mode [<признак>])
```

Аргумент: любое значение. nil отключает действие команды LAYERP (СЛОЙП), а T (и любое другое значение, отличное от nil) — включает. Возвращает новое состояние режима (T или nil).

Если аргумент <признак> опущен, то возвращается текущее состояние режима команды LAYERP (СЛОЙП).

Примеры:

- (acet-layerp-mode nil) отключает режим и возвращает nil
- (acet-layerp-mode) возвращает текущее состояние (nil или T)
- (acet-layerp-mode T) включает режим действия команды LAYERP (СЛОЙП) и возвращает T

1.18.17. Функция *acet-layerp-mark*

Устанавливает или снимает метку границы отката для команды LAYERP (СЛОЙП). В момент установки метки система запоминает текущее состояние слоев (для возможности возврата к нему).

```
(acet-layerp-mark [<флаг>])
```

Аргумент: любое значение. `nil` аннулирует метку границы отката (без метки команда LAYERP (СЛОЙП) не работает), а `T` (и любое значение, отличное от `nil`) устанавливает метку. Возвращает новый признак наличия метки отката (`T` или `nil`).

Если аргумент `<флаг>` опущен, то возвращается текущее состояние признака наличия метки отката команды LAYERP (СЛОЙП).

Примеры:

- `(acet-layerp-mark T)` устанавливает метку отката и возвращает `T`
- `(acet-layerp-mark)` возвращает текущее состояние признака (`nil` или `T`)
- `(acet-layerp-mark nil)` аннулирует метку конца отката команды LAYERP (СЛОЙП) и возвращает `nil`

1.18.18. Функция *acet-laytrans*

Выполняет трансляцию слоев.

```
(acet-laytrans <файл> [<параметры>])
```

Аргумент `<файл>` — DWG- или DWS-файл, из которого читается схема сопоставления слоев. Аргумент `<параметры>` — сумма установленных битовых флагов трансляции слоев. Состоит из следующих битовых флагов:

- 1 — установить для объектов цвет ByLayer (ПоСлою);
- 2 — установить для объектов тип линий ByLayer (ПоСлою);
- 4 — преобразовывать объекты в блоках;
- 8 — вести журнал транзакций.

Если второй аргумент опущен, то его значение принимается равным 15.

Возвращаемое значение — `T` при успешном завершении трансляции и `nil` — при ошибочном.

Пример:

- `(laytrans "D:\\DWS\\map.dws" 3)` выполняет трансляцию, используя схему из файла D:\DWS\map.dws, с преобразованием цветов и типов линий объектов слоев

1.18.19. Функция *acet-ms-to-ps*

Преобразует число из единиц длины пространства модели в единицы длины пространства листа применительно к видовому экрану. Не применяется при активизированной вкладке **Model** (Модель).

```
(acet-ms-to-ps [<число>] [<вэкран>])
```

Аргументы: <число> — целое или вещественное число, <вэкран> — примитив видового экрана, относительно которого преобразуется число. Если аргумент <число> не задан, то его значение запрашивается системой в интерактивном режиме. Если второй аргумент не задан, то в режиме **Model** (Модель) расчет выполняется с использованием масштаба активного экрана, а в режиме **Paper** (Лист) либо выбирается существующий видовой экран, если он только один, либо выдается запрос на выбор одного из нескольких видовых экранов текущей вкладки листа.

Возвращаемое значение — преобразованное вещественное число или `nil` (при обнаружении ошибки).

Пример:

- `(acet-ms-to-ps 5)` возвращает `0.1` для активного видового экрана с масштабом 0.02

1.18.20. Функция *acet-ps-to-ms*

Преобразует число из единиц длины пространства листа в единицы длины пространства модели применительно к выбранному видовому экрану. Не применяется при активизированной вкладке **Model** (Модель).

```
(acet-ps-to-ms [<число>] [<вэкран>])
```

Аргументы: <число> — целое или вещественное число, <вэкран> — примитив видового экрана, относительно которого преобразуется число. Если первый аргумент не задан, то число запрашивается системой в интерактивном режиме. Если аргумент <вэкран> не задан, то в режиме **Model** (Модель) расчет выполняется с использованием масштаба активного экрана, а в режиме **Paper** (Лист) либо выбирается существующий видовой экран, если он только один, либо выдается запрос на выбор одного из нескольких видовых экранов текущей вкладки листа.

Возвращаемое значение — преобразованное вещественное число или `nil` (при обнаружении ошибки).

Пример:

- `(acet-ps-to-ms 0.1)` возвращает `5.0` для активного видового экрана с масштабом 0.02

1.19. Файлы автоматической загрузки функций пользователя

В системе AutoCAD предусмотрены средства, обеспечивающие автоматическую загрузку LISP-программ при запуске сеанса редактирования системы AutoCAD, а также при открытии новых рисунков.

1.19.1. Файл acad.lsp

При запуске системы AutoCAD выполняется чтение и загрузка ряда файлов. В число автоматически загружаемых файлов входят файлы acad2006.lsp (в версии 2006) и acad.lsp, которые содержат выражения на языке LISP, в том числе определения необходимых функций.

Файл acad2006.lsp поставляется вместе с программным обеспечением системы AutoCAD 2006 (acad2005.lsp — с версией 2005) и содержит в виде исходных текстов ряд важных программ, которые по каким-то причинам не вошли в ядро системы. Редактировать этот файл ни в коем случае не рекомендуется.

Для загрузки программ пользователя предназначен специальный файл acad.lsp. Этот файл не входит в состав стандартного программного обеспечения системы AutoCAD, но при запуске AutoCAD пытается его найти и загрузить. Последовательность поиска файла acad.lsp определяется настройками путей поиска файлов поддержки. Первый же обнаруженный файл с названием acad.lsp загружается и дальнейший его поиск по другим папкам прекращается.

Если никакого файла acad.lsp в стандартных путях система не обнаружила, то это не является ошибкой и не препятствует дальнейшей работе.

Однако вы можете создать такой файл и включить в него любые выражения, которые будут выполнены в начале сеанса редактирования. Если в файле есть определения ваших функций, то они будут загружены в память первого рисунка. В листинге 1.32 приведен пример написания файла acad.lsp.

Листинг 1.32. Пример файла acad.lsp

```
; Глава 01\Листинг 1.32.lsp
(defun fkos1 (r1 pp1 / pp2)
  (command "_CIRCLE" pp1 r1) (setq vo (entlast))
  (initget 1) (setq pp2 (getpoint "\nЗадайте точку: "))
  (command "_COPY" vo "" pp1 pp2)
  (princ "\Окружности построены. ")(princ); тихий выход
); конец defun fkos1
(defun d2r (angdeg /) (* pi (/ angdeg 180.0)))
(setq curname "Design-Ritm")
(setvar "DIMSCALE" 50.0)
(princ "Выполнена загрузка моего файла acad.lsp. ")(princ)
```

В приведенном примере файл acad.lsp содержит определения двух пользовательских функций (`fkos1` и `d2r`), а три последние строки являются LISP-выражениями, которые будут выполнены при загрузке данного файла. С помощью этих выражений сформируется значение глобальной переменной `curname` и установится новое значение системной переменной DIMSCALE, равное 50.0.

При необходимости вы можете добавить в файл acad.lsp выражение, которое загрузит ваши функции из другого LSP-файла, например:

```
(load "d:\\User\\my_designtools.lsp")
```

Не рекомендуется включать в файл acad.lsp выражения, содержащие функции `command` и `vl-cmdf`, поскольку до полной инициализации среды команды системы AutoCAD не могут работать (см. функцию `s::startup`).

1.19.2. Файл acaddoc.lsp

Если файл acad.lsp загружается только один раз при запуске системы AutoCAD (об исключениях см. следующий раздел), то файлы acad2006doc.lsp и acaddoc.lsp версии 2006 загружаются при открытии каждого нового документа (рисунка).

В файл acad2006doc.lsp включены некоторые функции системы, не вошедшие в основное ядро, но которые нужно загрузить в память каждого рисунка. Аналогичную функцию выполняет файл acaddoc.lsp, но он предназначен для пользователя. В этом файле тоже могут быть исполняемые выражения и определения пользовательских функций.

Файл acaddoc.lsp загружается с каждым рисунком и только один раз. Поиск файла в путях настройки выполняется по тому же алгоритму, что и поиск файла acad.lsp. Первый же обнаруженный файл с таким именем загружается и дальнейший его поиск в данном рисунке прекращается.

1.19.3. Системная переменная ACADLSPASDOC

В обычном варианте настройки системы AutoCAD файл acad.lsp за весь сеанс редактирования загружается только один раз, даже если в этом сеансе открывалось много рисунков. На данный процесс оказывает влияние значение системной переменной ACADLSPASDOC.

Эта системная переменная может принимать только два значения: 0 (обычное значение) и 1. Последнее значение заставляет систему AutoCAD загружать файл acad.lsp с каждым открываемым рисунком, аналогично файлу acaddoc.lsp.

1.19.4. Функция *s::startup*

В системе AutoCAD предусмотрено одно имя для пользовательской функции (`s::startup`), которая, будучи записанной в файл acad.lsp или acaddoc.lsp, автоматически выполняется при открытии каждого рисунка после его полной инициализации.

ГЛАВА 2

Среда разработки приложений

Среда разработки приложений Visual LISP введена в систему AutoCAD для упрощения процесса написания и отладки программ. На рис. 2.1 показано диалоговое окно **Visual LISP for AutoCAD** (Visual LISP для AutoCAD), которое появляется на экране, если пользователь вводит в командной строке команду VLISP. Другой формой этой команды является VLIDE. К аналогичному результату приводит использование пункта меню **Tools | Visual LISP Editor | AutoLISP** (Сервис | Редактор Visual LISP | AutoLISP).

Диалоговое окно **Visual LISP for AutoCAD** (Visual LISP для AutoCAD) является окном специализированного приложения-оболочки, работающего внутри системы AutoCAD и имеющего свои меню и панели инструментов.

Внутри окна оболочки располагаются дочерние окна для открытых текстовых файлов (на рис. 2.1 показаны окна файлов imvector.dcl и imvector.lsp). Можно редактировать тексты файлов LISP-программ, DCL-файлов диалоговых окон, SQL-файлов операций с базами данных, файлов программ на языке C. Другие дочерние окна: **Visual LISP Console** (Консоль Visual LISP) — используется для связи с системой AutoCAD; **Trace** (Трассировка) — для отладки программ; **Build Output** (Сообщения сборки) — для вывода сообщений процесса построения исполняемых модулей.

2.1. Обзор меню

В состав среды Visual LISP входят девять меню: **File** (Файл), **Edit** (Правка), **Search** (Поиск), **View** (Вид), **Project** (Проект), **Debug** (Отладка), **Tools** (Сервис), **Window** (Окно), **Help** (Справка). Во всех меню есть как постоянные пункты, так и пункты, которые меняют свое состояние или исчезают в зависимости от конкретного этапа разработки программы.

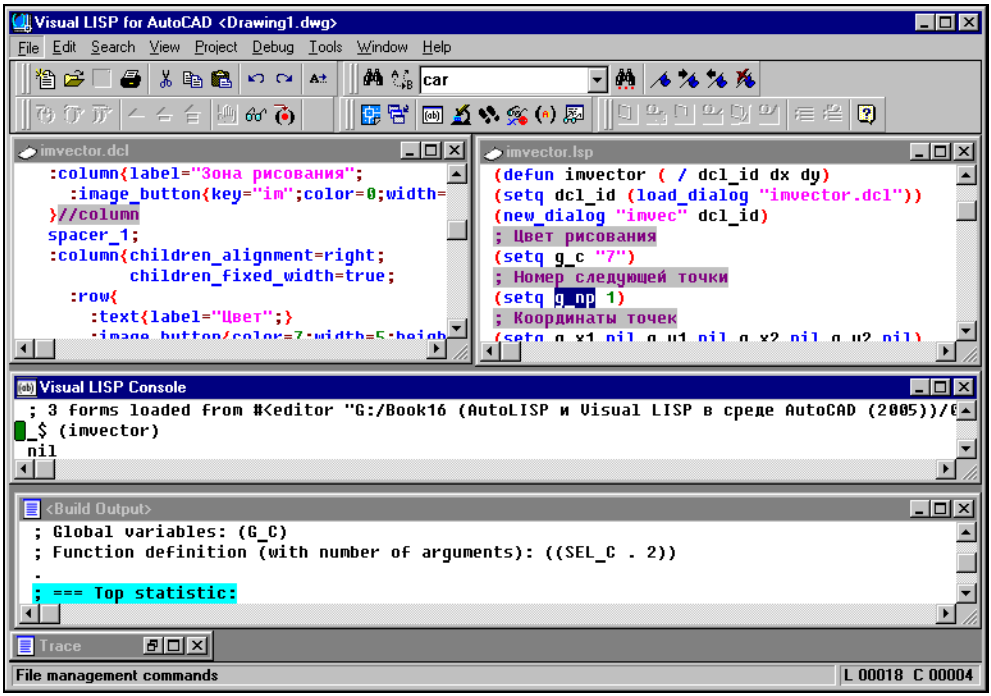


Рис. 2.1. Диалоговое окно **Visual LISP for AutoCAD**

Меню **File** (Файл) (рис. 2.2) предназначено для выполнения операций с файлами и включает в себя следующие пункты:

- **New File** (Новый файл);
- **Open File** (Открыть файл);
- **Reopen** (Открыть заново);
- **Save** (Сохранить);
- **Save As** (Сохранить как);
- **Save All** (Сохранить Все);
- **Close** (Закрыть);
- **Revert** (Обратить);
- **Close All** (Закрыть все);
- **Print** (Печать);
- **Print Setup** (Параметры печати);
- **Make Application** (Собрать приложение);
- **Load File** (Загрузить файл);
- **Exit** (Выход).

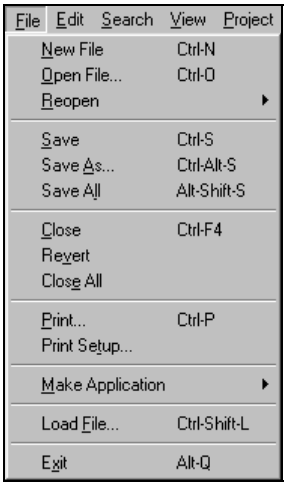


Рис. 2.2. Меню **File**

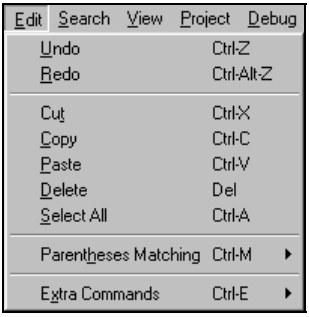


Рис. 2.3. Меню **Edit**

Назначение пунктов соответствует их наименованиям. Пункт **Revert** (Обратить) позволяет восстановить содержимое файла в активном окне аналогично пункту **Reopen** (Открыть заново). Действия, осуществляемые при выборе пункта **Make Application** (Собрать приложение), рассмотрены в *разд. 2.4.3*.

Меню **Edit** (Правка) (рис. 2.3) содержит следующие пункты:

- **Undo** (Отменить);
- **Redo** (Повторить);
- **Cut** (Вырезать);
- **Copy** (Копировать);
- **Paste** (Вставить);
- **Delete** (Удалить);
- **Select All** (Выделить все);
- **Parentheses Matching** (Проверка скобок);
- **Extra Commands** (Дополнительные функции).

Первые семь пунктов меню соответствуют стандартным пунктам меню **Edit** (Правка) других Windows-приложений. Пункт **Parentheses Matching** (Проверка скобок) содержит подпункты операций для контроля редактируемого текста:

- **Match Forward** (Найти закрывающую);
- **Match Backward** (Найти открывающую);
- **Select Forward** (Выделить до конца);
- **Select Backward** (Выделить до начала).

Два первых подпункта позволяют находить в тексте парную, закрывающую или открывающую скобку для скобки, находящейся у текущей позиции курсора. Два последних подпункта выполняют аналогичные операции, но при этом еще и выделяют необходимые участки текста.

Меню **Search** (Поиск) (рис. 2.4) предназначено для операций текстового редактора Visual LISP (в частности, операций поиска и замены) и включает в себя следующие пункты:

- **Find** (Найти);
- **Replace** (Заменить);
- **Find/Replace Next** (Найти/Заменить далее);
- **Complete Word by Match** (Дописать словом из окна);
- **Complete Word by Apropos** (Дописать словом из списка);
- **Bookmarks** (Закладки);
- **First message** (Первое сообщение);
- **Next message** (Следующее сообщение);
- **Go to Line** (Перейти на строку);
- **Go to Last Edited** (Последняя правка).

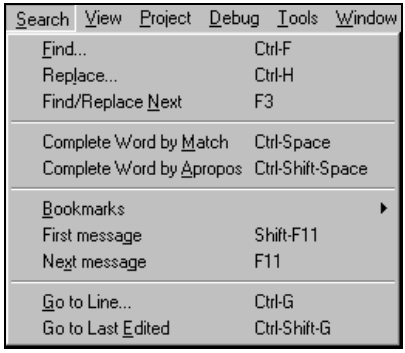


Рис. 2.4. Меню **Search**

Поясним назначение некоторых из этих пунктов (смысл остальных пунктов ясен из их названия). Пункт **Complete Word by Match** (Дописать словом из окна) предоставляет возможность редактору Visual LISP дописать (закончить) начатое вами слово аналогично предыдущему слову с таким же началом в текущем окне редактируемой программы. Пункт **Complete Word by Apropos** (Дописать словом из списка) позволяет получать варианты дописывания слова по его части (не обязательно начальной) из списка символов LISP, куда входят имена встроенных функций (рис. 2.5).

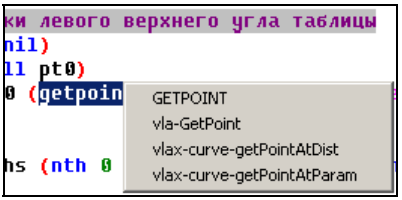


Рис. 2.5. Применение операции **Complete Word by Apropos**

С помощью закладок можно помечать нужные места в тексте для дальнейшего быстрого перемещения между ними. Операции установки, удаления закладок и переходов между закладками выполняются с помощью подпунктов **Toggle Bookmark** (Закладка Вкл/Откл), **Next Bookmark** (Следующая закладка), **Previous Bookmark** (Предыдущая закладка), **Clear All Bookmarks** (Убрать все закладки) пункта **Bookmarks** (Закладки). Еще два подпункта — **Select to Next Bookmark** (Выделить до следующей закладки) и **Select to Previous Bookmark** (Выделить до предыдущей закладки) — того же пункта меню позволяют облегчить операции выделения больших участков текста.

Меню **View** (Вид) (рис. 2.6) предназначено для операций просмотра вспомогательной информации во время работы LISP-программы, а также для работы с панелями инструментов диалогового окна **Visual LISP for AutoCAD** (Visual LISP для AutoCAD).

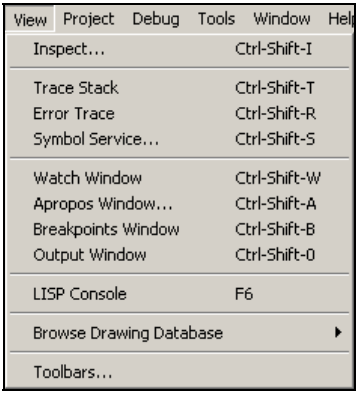


Рис. 2.6. Меню **View**

Меню состоит из следующих пунктов:

- **Inspect** (Изучить);
- **Trace Stack** (Стек трассировки);
- **Error Trace** (Поиск ошибок);
- **Symbol Service** (Инспектор символов);

- **Watch Window** (Окно контрольных значений);
- **Apropos Window** (Параметры справочного списка);
- **Breakpoints Window** (Окно точек останова);
- **Output Window** (Окно вывода);
- **LISP Console** (Консоль LISP);
- **Browse Drawing Database** (Обзор базы данных рисунка);
- **Toolbars** (Панели инструментов).

Пункт **Inspect** (Изучить) позволяет вызывать специальные окна **Inspect** (Изучение), в которых можно посмотреть состояние переменных, списков, наборов, примитивов, а также выражений LISP. Имя изучаемого объекта или выражение должно быть предварительно выделено в тексте программы; если ничего не было выделено, перед открытием очередного окна **Inspect** (Изучение) выдается запрос имени. Пример открытого окна приведен на рис. 2.7, где показаны элементы списка с данными полилинии.

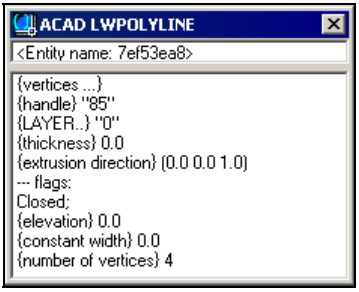


Рис. 2.7. Окно **Inspect**

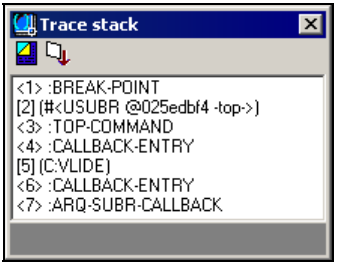


Рис. 2.8. Окно **Trace stack**

Двойной щелчок по любой строке с элементом списка вызывает окно **Inspect** (Изучение) для этого элемента.

Пункт **View | Trace Stack** (Вид | Стек трассировки) (см. рис. 2.6) выводит на экран окно **Trace stack** (Стек трассировки). В этом окне приводится перечень последних выполненных операций (рис. 2.8).

В верхней части окна расположены кнопки:

- — очищает содержимое окна;
- — копирует текст из окна **Trace stack** (Стек трассировки) в специальное окно **Trace** (Трассировка).

Если в окне **Trace stack** (Стек трассировки) выделить какую-нибудь строку, то с помощью правой кнопки мыши можно вызвать контекстное меню с дополнительными операциями над элементами стека трассировки (объем контекстного меню может меняться): **Inspect** (Изучить), **Print** (Печать), **Function Symbol**

(Символ функции), **Copy** (Копировать), **Local Variables** (Локальные переменные), **Source Position** (Положение в тексте), **Call Point Source** (Позиция вызова функции).

Пункт меню **View | Error Trace** (Вид | Поиск ошибок) вызывает окно **Error trace** (Поиск ошибок) (рис. 2.9) с информацией об ошибках, возникших при выполнении программы.

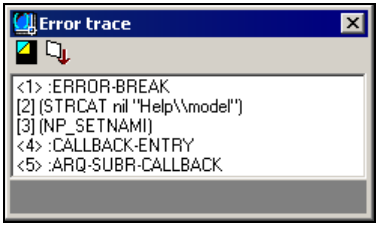


Рис. 2.9. Окно **Error trace**

На рис. 2.9 видно, что при вычислении выражения `(STRCAT nil "Help\\model")`, приведенного во второй строке сверху, была обнаружена ошибка. В чем заключается ошибка — должен решить программист (в данном случае — недопустимый тип первого аргумента, переданного функции `strcat`).

Если выделить в этом окне строку, то по нажатии правой кнопки мыши можно вызвать контекстное меню для получения дополнительных сведений об элементе, отраженном в окне **Error trace** (Поиск ошибок). Контекстное меню имеет те же пункты, что и контекстное меню окна **Trace stack** (Стек трассировки).

Пункт **Symbol Service** (Инспектор символов) меню **View** (Вид) открывает одноименное окно (рис. 2.10), которое выполняет функции, похожие на функции окна **Inspect** (Изучение), но только для символов языка LISP.

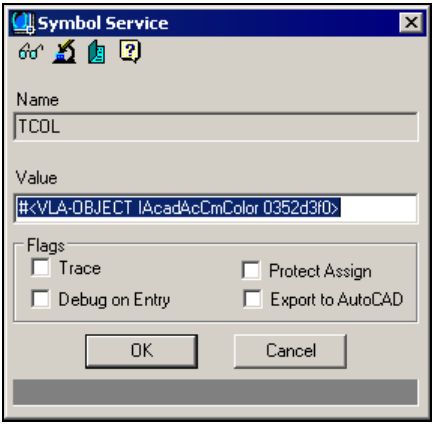


Рис. 2.10. Окно **Symbol Service**

Окно имеет поля **Name** (Имя) и **Value** (Значение), а также область **Flags** (Флаги). Назначение первых двух полей очевидно (имя символа и значение символа), а в области **Flags** (Флаги) расположены четыре флажка, которые могут быть установлены для выбранного символа:

- **Trace** (Трассировка) — осуществляет отслеживание с помощью трассировки;
- **Debug on Entry** (Отладка на входе) — выполняет отладку при входе в функцию;
- **Protect Assign** (Защита от присвоения) — выполняет защиту специальных символов от случайного присвоения (например, `T`, `pi` и др.);
- **Export to AutoCAD** (Экспорт в AutoCAD) — экспортирует функцию, чтобы сделать ее доступной для приложений ObjectARX.

Окно **Symbol Service** (Инспектор символов) в верхней его части имеет четыре кнопки (слева направо):

- — добавляет символ к окну **Watch** (Контрольное значение);
- — открывает окно **Inspect** (Изучение) для данного символа;
- — вызывает окно текстового редактора и выделяет в нем текст символа, являющегося пользовательской функцией;
- — вызывает справку по языку Visual LISP с описанием символа, являющегося встроенной функцией.

Пункт меню **View | Watch Window** (Вид | Окно контрольных значений) выводит на экран окно **Watch** (Контрольное значение) (рис. 2.11).

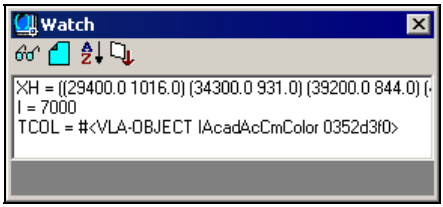


Рис. 2.11. Окно **Watch**

Окно отображает значения выбранных символов в ходе выполнения программы. Добавление символов к этому окну производится либо посредством кнопки окна **Symbol Service** (Инспектор символов), либо с помощью такой же кнопки (левой) из ряда кнопок, расположенных в верхней части окна **Watch** (Контрольное значение). Перечислим эти кнопки и их функции:

- — добавляет контрольное значение;
- — очищает окно;
- — сортирует символы и выражения по алфавиту;
- — копирует данные в окно трассировки.

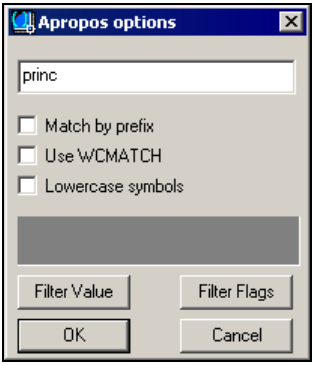


Рис. 2.12. Окно **Apropos options**

Пункт меню **View | Apropos Window** (Вид | Параметры справочного списка) выводит на экран окно **Apropos options** (Параметры справочного списка) (рис. 2.12).

Данное окно дает возможность просматривать список зарезервированных символов (в первую очередь, имен встроенных функций). Выделенная заранее в тексте программы или введенная в верхнем поле последовательность букв (на рис. 2.12 введены символы `princ`) является основным критерием формирования списка выводимых имен (эти буквы будут присутствовать в именах всех символов списка). Пользователь может с помощью флажка **Match by prefix** (По префиксу) указать, должна ли эта последовательность обязательно присутствовать в начале имени (для этого необходимо установить флажок) или может встретиться в другой части символа (для этого флажок следует сбросить), а также возможно ли использование специальных знаков типа "*" или "#", как в функции `wcmatch`. На рис. 2.13 приведено результирующее окно **Apropos results** (Справочный список найденных символов) с пятью символами LISP, включающими `princ`.

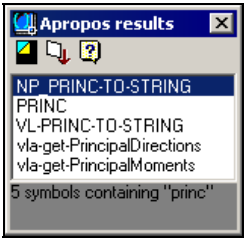


Рис. 2.13. Окно **Apropos results**

Пункт **View | Breakpoints Window** (Вид | Окно точек останова) открывает окно **Breakpoints** (Точки останова) (рис. 2.14), используемое при отладке программ.

Пункт **Output Window** (Окно вывода) позволяет открыть окно **<Builder Ouput>** (<Сообщения сборки>) для вывода сообщений, генерируемых в процессе сборки приложений.

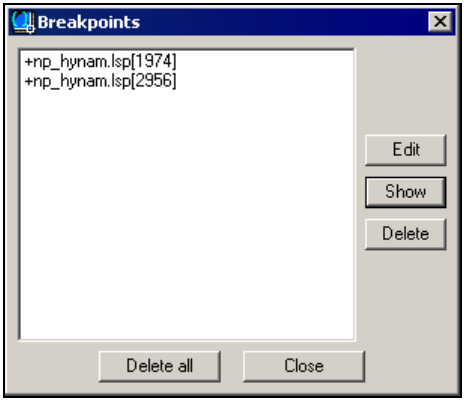


Рис. 2.14. Окно **Breakpoints**

Пункт **View | LISP Console** (Вид | Консоль LISP) вызывает очень важное окно, называемое **Visual LISP Console** (Консоль Visual LISP) (рис. 2.15), которое в среде Visual LISP является аналогом зоны командных строк системы AutoCAD. В нем можно вводить выражения языка LISP, загружать и выполнять функции, а также читать значения переменных (для последней операции нужно в текущей строке ввести имя переменной и нажать клавишу <Enter>; восклицательный знак перед именем переменной в таком случае не требуется). Сюда же Visual LISP выводит свои сообщения.

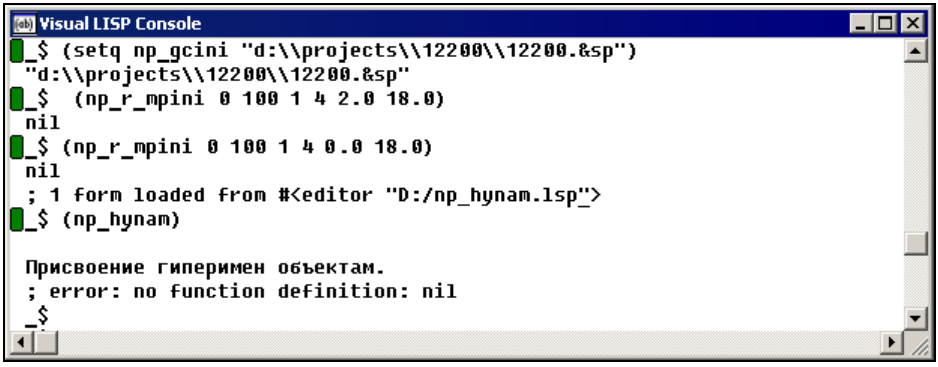


Рис. 2.15. Окно **Visual LISP Console**

Подменю **View | Browse Drawing Database** (Вид | Обзор базы данных рисунка) имеет пункты **Browse All Entities** (Все объекты), **Browse Tables** (Таблицы), **Browse Blocks** (Блоки), **Browse Selection** (Набор) и **Inquire Extended Data** (Расширенные данные). При выборе первых четырех пунктов вызываются соответствующие окна для просмотра базы данных графических примитивов и неграфических объектов текущего рисунка (рис. 2.16).

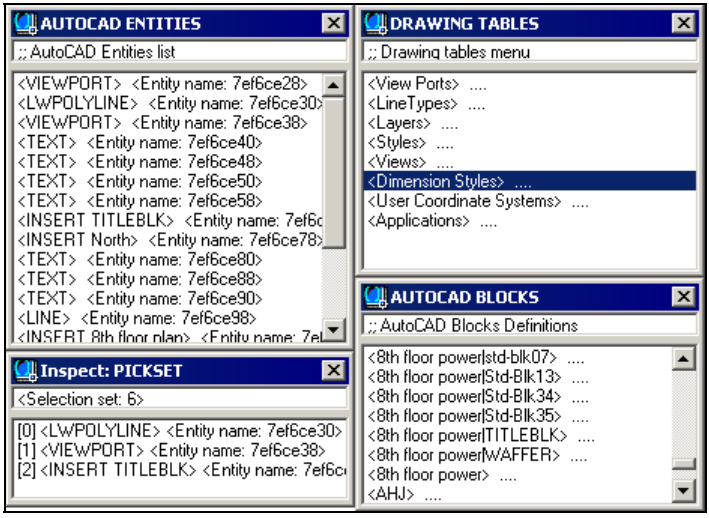


Рис. 2.16. Окна обзора базы данных рисунка

Пункт **Inquire Extended Data** (Расширенные данные) предназначен для просмотра расширенных данных примитива. Поскольку у примитива могут быть расширенные данные от нескольких приложений, после выбора указанного подпункта сначала открывается окно **Choose Applications to Show** (Выбор приложений для просмотра) (рис. 2.17).

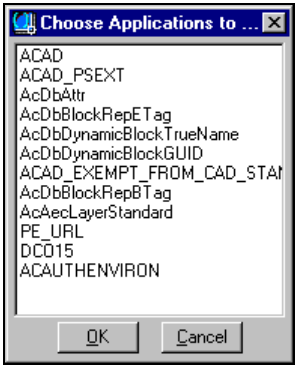


Рис. 2.17. Окно **Choose Applications to Show**

В этом окне можно отметить одну или несколько строк с именами тех приложений, чьи расширенные данные интересуют пользователя. Далее необходимо выбрать подпункт **Browse Selection** (Набор) и по запросу отметить тот примитив, данные которого будут просматриваться. Затем в появившемся окне **Inspect: PICKSET** (Изучение: НАБОР) двойной щелчок по строке с именем примитива

вызовет окно с данными этого примитива. В появившемся окне нужно вызвать контекстное меню (рис. 2.18).

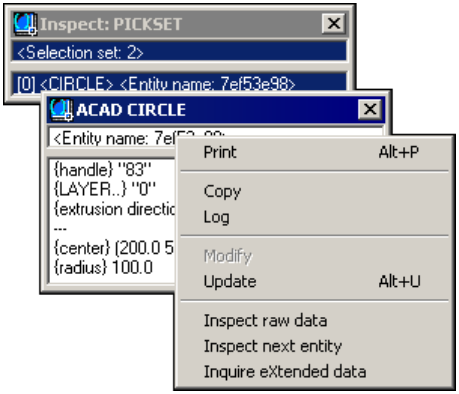


Рис. 2.18. Контекстное меню операций с данными примитива, вызванное из окна **Inspect: PICKSET**

Теперь выбор пункта **Inspect raw data** (Изучить необработанные данные) выведет окно **Inspect: LIST** (Изучение: СПИСОК) (рис. 2.19), в котором в нижней строке будут присутствовать расширенные данные отмеченного графического объекта для выбранных приложений.

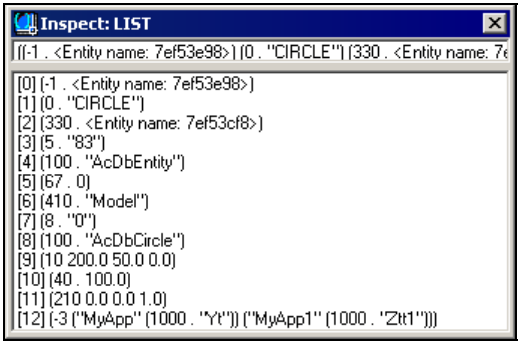


Рис. 2.19. Окно **Inspect: LIST** с расширенными данными

Пункт меню **View | Toolbars** (Вид | Панели инструментов) вызывает диалоговое окно **Toolbars** (Панели инструментов) (рис. 2.20), которое управляет видимостью пяти панелей среды Visual LISP.

Флажки этого окна должны быть установлены около имен тех из пяти панелей (**Standard** (Стандартная), **Search** (Поиск), **Tools** (Сервис), **Debug** (Отладка), **View** (Вид)), которые необходимо вывести на экран. Изменения вносятся с помощью кнопок **Hide All** (Скрыть все), **Show All** (Показать все), **Apply** (Применить).

Кнопка **Close** (Закрыть) закрывает диалоговое окно **Toolbars** (Панели инструментов).

Меню **Project** (Проект) (рис. 2.21) дает возможность работы с проектами.

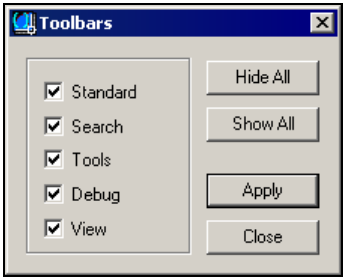


Рис. 2.20. Диалоговое окно **Toolbars**

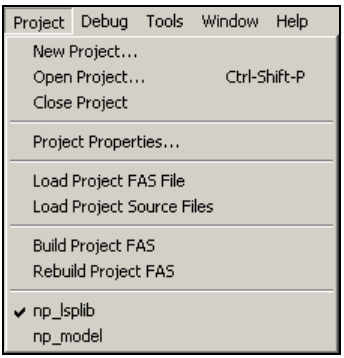


Рис. 2.21. Меню **Project**

Проект — это именованное объединение LSP-файлов, которое позволяет создать для них один исполняемый файл с расширением fas (можно для каждого из LSP-файлов проекта получать свой FAS-файл, однако такой вариант менее удобен). Меню **Project** (Проект) имеет следующие пункты:

- **New Project** (Новый проект);
- **Open Project** (Открыть проект);
- **Close Project** (Закрыть проект);
- **Project Properties** (Свойства проекта);
- **Load Project FAS File** (Загрузить FAS-файл проекта);
- **Load Project Source Files** (Загрузить исходные файлы проекта);
- **Build Project FAS** (Собрать FAS-файл проекта);
- **Rebuild Project FAS** (Перекомпилировать FAS-файл проекта).

Все перечисленные пункты рассмотрены в *разд. 2.4.2*. Последние (дополнительные) строки в меню показывают имена открытых проектов. Активный проект помечен флажком.

Меню **Debug** (Отладка) (рис. 2.22) позволяет отлаживать программы в среде Visual LISP.

Это меню имеет такие пункты:

- **Step Into** (Шаг с заходом);
- **Step Over** (Шаг с обходом);
- **Step Out** (Шаг с выходом);

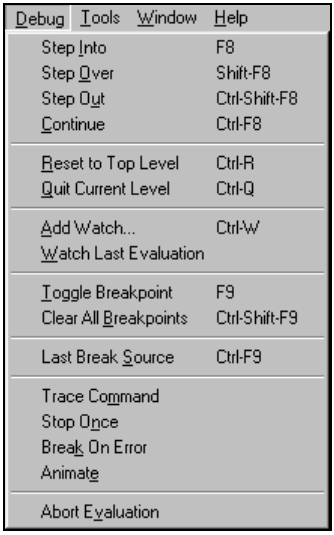


Рис. 2.22. Меню **Debug**

- **Continue** (Продолжить);
- **Reset to Top Level** (Сброс до верхнего уровня);
- **Quit Current Level** (Выход из текущего уровня);
- **Add Watch** (Добавить контрольное значение);
- **Watch Last Evaluation** (Результат последнего вычисления);
- **Toggle Breakpoint** (Точка останова Вкл/Откл);
- **Clear All Breakpoints** (Удалить все точки останова);
- **Last Break Source** (Причина последнего останова);
- **Trace Command** (Команда трассировки);
- **Stop Once** (Останов);
- **Break On Error** (Останов на ошибке);
- **Animate** (Анимация);
- **Abort Evaluation** (Прервать вычисления).

Эти пункты и соответствующие им операции рассмотрены далее в *разд. 2.4.1.*

Меню **Tools** (Сервис) (рис. 2.23) предназначено для вспомогательных операций.

Если в среде Visual LISP активно окно **Visual LISP Console** (Консоль Visual LISP), то меню **Tools** (Сервис) имеет следующие пункты и подменю:

- **AutoCAD Mode** (Режим AutoCAD);
- **Window Attributes** (Оформление окна);

- **Environment Options** (Параметры среды);
- **Save Settings** (Сохранить параметры).

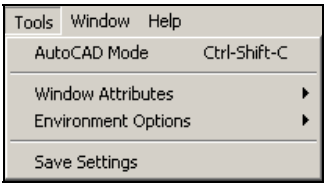


Рис. 2.23. Меню **Tools** (активно окно **Visual LISP Console**)

Пункт **AutoCAD Mode** (Режим AutoCAD) включает или отключает режим использования окна **Visual LISP Console** (Консоль Visual LISP) в качестве командной строки системы AutoCAD (появляется подсказка ***Command:*** *(Команда:)*). Это дает возможность вводить команды AutoCAD прямо в данном окне. Если такой режим установить, то в меню слева от пункта **AutoCAD Mode** (Режим AutoCAD) появится флажок.

Подменю **Window Attributes** (Оформление окна) состоит из пяти пунктов:

- **Syntax Coloring** (Цвета синтаксических элементов);
- **Configure Current** (Настроить текущее);
- **Set Current Window to Prototype** (Оформление текущего окна по прототипу);
- **Set All Windows to Prototype** (Оформление всех окон по прототипу);
- **Font** (Шрифт).

Пункт **Syntax Coloring** (Цвета синтаксических элементов) этого подменю доступен только в том случае, если в активном окне находится текст файла, имеющего любое расширение, кроме lsp. Тогда вызывается окно **Color Style** (Цветовой стиль) (на рис. 2.24 показано окно, открытое при работе с DCL-файлом), в котором можно выбрать цветовую гамму текстового редактора, отвечающую одному из предлагаемых вариантов. Подробнее о цветовой гамме текстового редактора рассказано в *разд. 2.3.*

Пункт **Configure Current** (Настроить текущее) предназначен для вызова окна **Window Attributes** (Оформление окна) (рис. 2.25), в котором можно подкорректировать действующие цветовые установки разных элементов текста. Оформляемые элементы перечислены в раскрывающемся списке.

Пункт **Set Current Window to Prototype** (Оформление текущего окна по прототипу) меняет установки активного окна на установки прототипа, а подпункт **Set All Windows to Prototype** (Оформление всех окон по прототипу) выполняет ту же операцию над всеми окнами.

Рис. 2.24. Окно **Color Style**

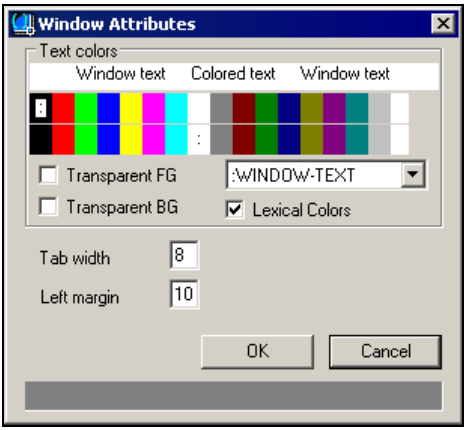


Рис. 2.25. Окно **Window Attributes**

Пункт подменю **Configure Current | Font** (Настроить текущее | Шрифт) вызывает стандартное диалоговое окно **Select Font** (Выбор шрифта) для выбора используемых в окнах шрифтов.

Подменю **Tools | Environment Options** (Сервис | Параметры среды) имеет три пункта:

- **General Options** (Общие параметры);
- **Visual LISP Format Options** (Параметры форматирования Visual LISP);
- **Page Setup Options** (Параметры разметки страницы).

Пункт **General Options** (Общие параметры) вызывает диалоговое окно **General Options** (Общие параметры), имеющее вкладки **General** (Общие) (рис. 2.26) и **Diagnostic** (Диагностические) (рис. 2.27), позволяющие менять различные настройки среды Visual LISP.

Пункт **Visual LISP Format Options** (Параметры форматирования Visual LISP) раскрывает диалоговое окно **Format options** (Опции форматирования) (рис. 2.28), управляющее установками форматирования для текстового редактора Visual LISP.

Третий пункт — **Page Setup Options** (Параметры разметки страницы) подменю **Environment Options** (Параметры среды) вызывает диалоговое окно **Page Setup** (Параметры страницы) для операций вывода текстов на печать из среды Visual LISP.

Пункт меню **Tools | Save Settings** (Сервис | Сохранить параметры) позволяет сохранять настройки рабочего стола Visual LISP (последние открытые окна, их параметры и т. п.). Можно задать автоматическое сохранение этих настроек, установив флажок **Autosave setting on exit application** (Автосохранение при выходе из приложения) в области **DESKTOP** (РАБОЧИЙ СТОЛ) вкладки **General** (Общие) диалогового окна **General Options** (Общие параметры) (см. рис. 2.26).

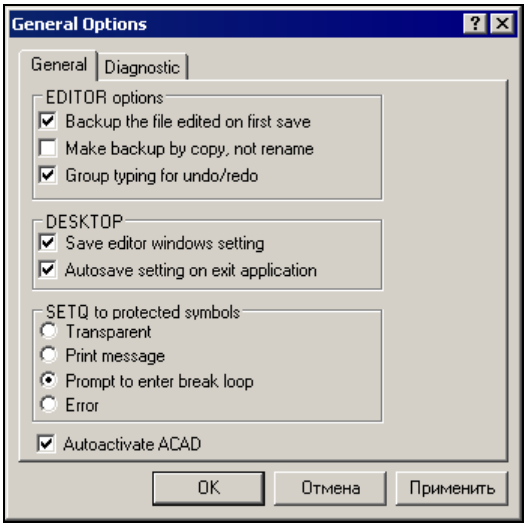


Рис. 2.26. Диалоговое окно **General Options**, вкладка **General**

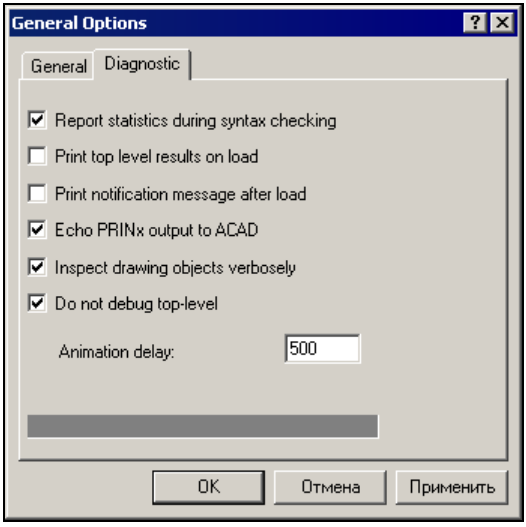


Рис. 2.27. Диалоговое окно **General Options**, вкладка **Diagnostic**

Если в среде Visual LISP активно окно с текстом редактируемого файла программы, то в меню **Tools** (Сервис) (рис. 2.29) присутствуют следующие дополнительные пункты и подменю:

- **Load Selection** (Загрузить выделенный фрагмент);
- **Load Text in Editor** (Загрузка текста в редактор);

- **Check Selection** (Проверить выделенный фрагмент);
- **Check Text in Editor** (Проверить текст в редакторе);
- **Format code in Selection** (Форматировать во фрагменте);
- **Format code in Editor** (Форматировать код в редакторе);
- **Interface Tools** (Инструменты интерфейса).

Первые шесть перечисленных пунктов позволяют загружать для выполнения, проверять или форматировать как текст всей программы, так и ее выделенный

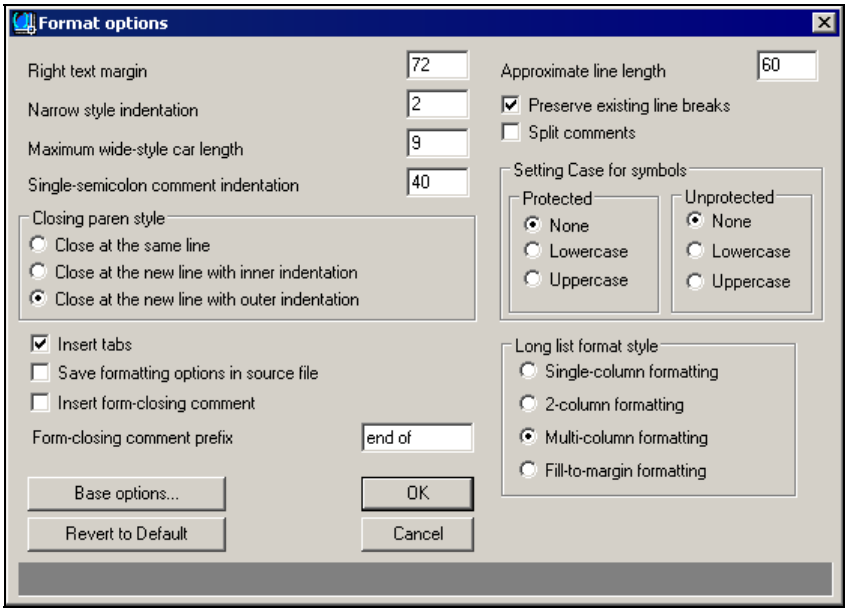


Рис. 2.28. Диалоговое окно **Format options**

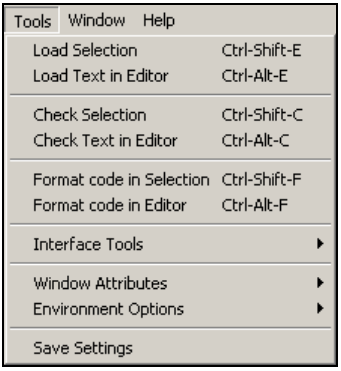


Рис. 2.29. Меню **Tools** (активно окно LSP-файла)

фрагмент. Работа этих пунктов рассмотрена далее в *разд. 2.3* и *2.4.1*. Подменю **Interface Tools** (Инструменты интерфейса) имеет два подпункта:

- **Preview DCL in Selection** (Просмотр DCL из фрагмента);
- **Preview DCL in Editor** (Просмотр DCL из редактора).

Эти подпункты позволяют загружать диалоговое окно из редактируемого DCL-файла или его выделенной части, что является удобным способом предварительной проверки синтаксиса диалогового окна.

Меню **Window** (Окно) (рис. 2.30) по структуре аналогично меню с подобным наименованием других приложений Windows.

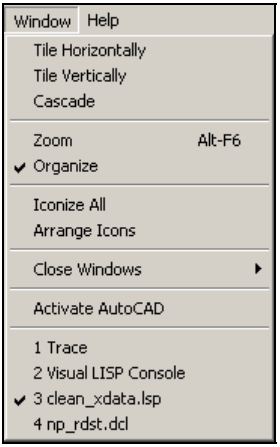


Рис. 2.30. Меню **Window**

Оно позволяет управлять взаимным расположением нескольких одновременно открытых окон в среде Visual LISP. Пункт **Activate AutoCAD** (Активизировать AutoCAD) данного меню сворачивает диалоговое окно **Visual LISP for AutoCAD** (Visual LISP для AutoCAD) и делает временно доступным основное окно системы AutoCAD (например, для выполнения в рисунке каких-нибудь построений). В нижней части меню **Window** (Окно) приводится список открытых окон, при этом флажком помечается активное окно.

Последнее меню **Help** (Справка) (рис. 2.31) состоит из двух пунктов:

- **Visual LISP Help Topics** (Справка о Visual LISP) — открывает разделы справочной системы, посвященные разработке приложений на языках LISP и VBA;

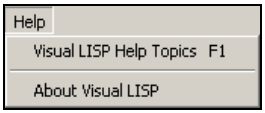


Рис. 2.31. Меню **Help**

- **About Visual LISP** (О программе Visual LISP) — открывает окно с информацией о разработчиках среды Visual LISP.

2.2. Панели инструментов Visual LISP

Среда Visual LISP имеет пять панелей инструментов (рис. 2.32—2.36):

- **Standard** (Стандартная);
- **Search** (Поиск);
- **Tools** (Сервис);
- **Debug** (Отладка);
- **View** (Вид).

Рис. 2.32. Панель инструментов **Standard**

Рис. 2.33. Панель инструментов **Search**

Рис. 2.34. Панель инструментов **Tools**

Рис. 2.35. Панель инструментов **Debug**

Рис. 2.36. Панель инструментов **View**

Наименования панелей похожи на наименования рассмотренных в предыдущем разделе меню. Функции панелей также аналогичны функциям соответствующих меню. Назначение кнопок (все они имеют подсказки) соответствует назначению пунктов меню, рассмотренных ранее.

2.3. Написание и проверка синтаксиса программ в текстовом редакторе среды Visual LISP

В среде Visual LISP можно открывать в своих окнах и редактировать текстовые файлы следующих типов: LISP-программ (с расширением lsp), DCL-файлы диалоговых окон (с расширением dcl), SQL-файлы операций с базами данных (с расширением sql), файлы программ на языке C++ (с расширениями c, cpp). Включенный в состав среды текстовый редактор оказывает существенную помощь в написании файлов этих типов, поскольку уже в процессе ввода текста он выполняет синтаксический анализ, выделяя символы, соответствующие скобкам, операторам, операндам, комментариям и т. д. Результаты своего синтаксического анализа редактор отображает в набранном тексте цветом букв. Выбор настройки редактора в соответствии с типом файла можно сделать с помощью пункта меню **Tools | Window Attributes | Syntax Coloring** (Сервис | Оформление окна | Цвета синтаксических элементов). По умолчанию редактор настроен на работу с текстами LISP-программ.

В редакторе можно открывать и другие текстовые файлы — например, файлы шаблонов меню (с расширением mnu), файлы адаптаций (с расширением cui), текстовые файлы (с расширением txt), но в этих файлах синтаксический анализ не производится.

Цвета, обычно используемые для синтаксического анализа программ на языке LISP, следующие:

- синий — зарезервированные символы LISP (имена встроенных и внешних функций и т. п.);
- фиолетовый — текстовые строки, обрамленные двойными кавычками;
- зеленый — целые числа;
- темно-голубой (более похож на зеленый) — вещественные числа;
- фиолетовый (на сером фоне) — комментарии;
- красный — открывающие и закрывающие скобки;
- черный — прочие символы (символы пользователя и т. д.).

При желании можно внести изменения в цветовую палитру с помощью пункта меню **Tools | Window Attributes | Configure Current** (Сервис | Оформление окна | Настроить текущее).

Подменю **Environment Options** (Параметры среды) меню **Tools** (Сервис) тоже может помочь в адаптации среды под конкретного пользователя.

Незаменимым средством быстрого комментирования выделенного участка текста и снятия комментариев являются кнопки и панели инструментов **Tools** (Сервис).

Синтаксическая проверка текста может быть выполнена с помощью кнопок (проверяет весь файл в активном окне) и (проверяет выделенный участок текста в активном окне) панели **Tools** (Сервис). Выделение участка текста выполняется путем протаскивания мыши с нажатой левой кнопкой.

Для проверки соответствия закрывающих и открывающих скобок хорошо пользоваться пунктом **Parentheses Matching** (Проверка скобок) меню **Edit** (Правка).

Замечание

При проверке синтаксиса программы можно получать важную информацию о количестве аргументов функций и именах глобальных переменных проверяемого файла. Для этого на вкладке **Diagnostics** (Диагностические) диалогового окна **General Options** (Общие параметры) необходимо установить флажок **Report statistics during syntax checking** (Вывод статистики в ходе синтаксической проверки) (см. рис. 2.27).

При перезаписи файла с расширением lsp Visual LISP создает копию предыдущего состояния файла (копии присваивается расширение _ls).

2.4. Разработка приложений на языке LISP

С помощью Visual LISP удобно писать не отдельные программы, а целые приложения. Приложения только из компилированных LSP-файлов (FAS-приложения) можно создавать с помощью механизма проектов, рассмотренного в *разд. 2.4.2*. Если в приложение необходимо включить DCL-файлы диалоговых окон или вспомогательные текстовые файлы, то необходимо пользоваться инструментом разработки сложных приложений (VLX-приложений, *см. разд. 2.4.3*).

2.4.1. Средства отладки программ

Специальное окно **Visual LISP Console** (Консоль Visual LISP) заменяет пользователю в среде Visual LISP зону командных строк системы AutoCAD. С консоли можно вводить выражения на языке LISP, загружать и выполнять функции, в него же выводятся сообщения Visual LISP и системы AutoCAD.

Для загрузки LISP-программы (аналогично загрузке файла с помощью функции `load` языка LISP), находящейся в открытом окне текстового редактора, следует применить пункт меню **Tools | Load Text in Editor** (Сервис | Загрузка текста из редактора) или кнопку панели инструментов **Tools** (Сервис). На рис. 2.37 показано сообщение, генерируемое Visual LISP о загрузке файла.

Возможно пошаговое выполнение программы, когда пользователь поочередно выделяет и загружает фрагменты текста (используется пункт меню **Tools | Load Selection** (Сервис | Загрузить выделенный фрагмент) или кнопка панели инструментов **Tools** (Сервис)).

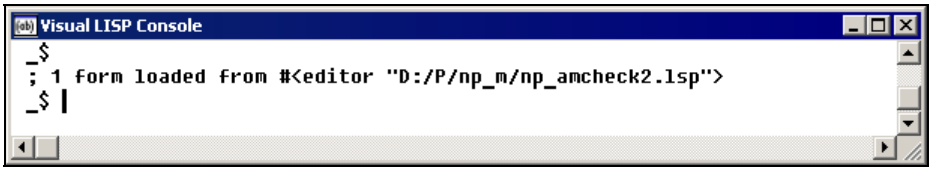


Рис. 2.37. Сообщение о загрузке файла

В окне **Visual LISP Console** (Консоль Visual LISP) в качестве приглашения пользователя к вводу применяются символы "_$". Так, например, если пользователь загрузил файл rcl.lsp с одноименной функцией, он может выполнить ее, введя в консоли `(rcl)`. Visual LISP выполнит программу и, если в ней нет ошибок, выведет результат на консоль.

Прямо с консоли можно читать значения глобальных переменных. Для этого нужно в ответ на приглашение ввести с консоли имя переменной. Visual LISP выведет ее текущее значение (на рис. 2.38 показано значение переменной `e0`). Другой способ прочитать значение — выделить в тексте имя переменной (например, двойным щелчком на имени) и нажать кнопку .

Рис. 2.38. Чтение значения глобальной переменной в окне **Visual LISP Console**

Замечание

В среде Visual LISP при чтении значения переменной не требуется ввод перед именем переменной восклицательного знака, как это необходимо в системе AutoCAD *(см. разд. 1.1)*.

При отладке программы удобно следить за значениями выбранных глобальных переменных. Для этого используются следующие пункты меню **View** (Вид): **Inspect** (Изучить), **Symbol Service** (Инспектор символов), **Watch Window** (Окно контрольных символов). Вызов при помощи указанных пунктов соответствующих окон (см. рис. 2.7, 2.10 и 2.11) показывает текущие значения выбранных переменных или выражений. Содержимое этих окон может быть скопировано в окно **Trace** (Трассировка) для последующего обобщения.

При наличии ошибок следует также пользоваться пунктами **Trace Stack** (Стек трассировки) и **Error Trace** (Поиск ошибок), которые выводят соответствующие одноименные окна (см. рис. 2.8 и 2.9).

Действенным средством отладки является расстановка в программе точек ее прерывания (останова). Это выполняется с помощью пункта меню **Debug | Toggle**

Breakpoint (Отладка | Точка останова Вкл/Откл) или кнопки панели инструментов **Debug** (Отладка). В таком случае программа при своем выполнении, дойдя до точки прерывания, останавливается и ждет указания от пользователя о возможном варианте продолжения работы. Для продолжения работы предназначены пункты **Step Into** (Шаг с заходом), **Step Over** (Шаг с обходом), **Step Out** (Шаг с выходом), **Continue** (Продолжить), **Reset to Top Level** (Сброс до верхнего уровня), **Quit Current Level** (Выход из текущего уровня) меню **Debug** (Отладка) или аналогичные им кнопки панели инструментов **Debug** (Отладка).

Особенно эффективно сочетание точек прерывания с открытым окном контрольных значений (см. рис. 2.11). Выполняя свою программу по частям, вы будете следить за тем, как при этом изменяются значения контрольных переменных.

Пункт меню **Debug | Animation** (Отладка | Анимация) дает возможность следить за ходом выполнения программы по исходному тексту, т. к. Visual LISP поочередно выделяет те выражения программы, которые обрабатываются в текущий момент времени. Интервал между шагами анимации задается в поле **Animation delay** (Задержка анимации) на вкладке **Diagnostic** (Диагностические) окна **General Options** (Общие параметры) (см. рис. 2.27).

Для отключения анимации нужно снять флажок слева от пункта **Animate** (Анимация) в меню **Debug** (Отладка).

2.4.2. Проекты и FAS-приложения

Несколько LISP-программ, выполняемых совместно, можно объединять в проекты. Каждый такой проект является самостоятельным приложением, которое может работать в исходных текстах или в компилированных файлах с расширением fas. Кроме того, проект можно скомпилировать в единый FAS-файл, что даст преимущества по скорости загрузки и выполнения.

Операции с проектами собраны в меню **Project** (Проект). Для создания нового проекта используется пункт **New Project** (Новый проект), который открывает диалоговое окно **New project** (Новый проект) (рис. 2.39).

В этом окне нужно ввести имя файла с данными о проекте (расширение файла — prj). В таком файле Visual LISP сохраняет информацию о входящих в проект программных файлах и о режимах построения исполняемого модуля приложения. После ввода имени проекта появляется окно **Project properties** (Свойства проекта) (рис. 2.40).

Диалоговое окно имеет две вкладки: **Project Files** (Файлы проекта) (см. рис. 2.40) и **Build Options** (Параметры сборки) (рис. 2.41). В первой из них нужно в поле **Look in** (Искать в) выбрать папку с LSP-файлами и в появившемся слева списке обнаруженных файлов с помощью кнопки > перенести нужные имена в правую часть. Если файлы проекта находятся в разных папках, можно поменять имя папки и выбрать из нее следующую часть включаемых файлов. Для файлов из разных папок с помощью щелчка правой кнопкой мыши по имени файла можно вы-

звать контекстное меню, где выбор пункта **Log filenames and size** (Имена и размеры файлов) выводит в нижнюю часть диалогового окна для каждого выделенного имени его полное имя и размер.

По окончании формирования полного списка компонентов нового проекта можно их упорядочить с помощью кнопок **Top** (Начало), **Up** (Вверх), **Down** (Вниз),

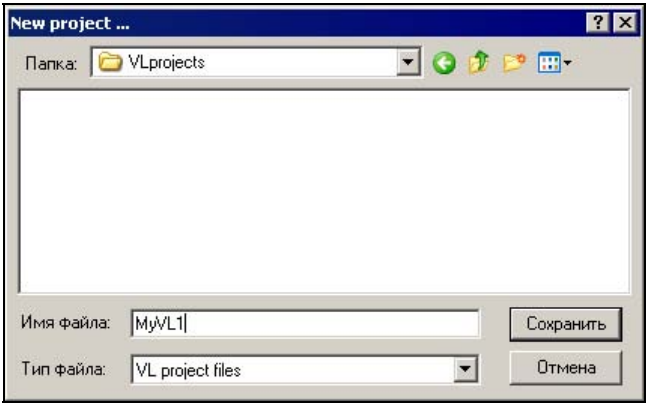


Рис. 2.39. Диалоговое окно **New project**

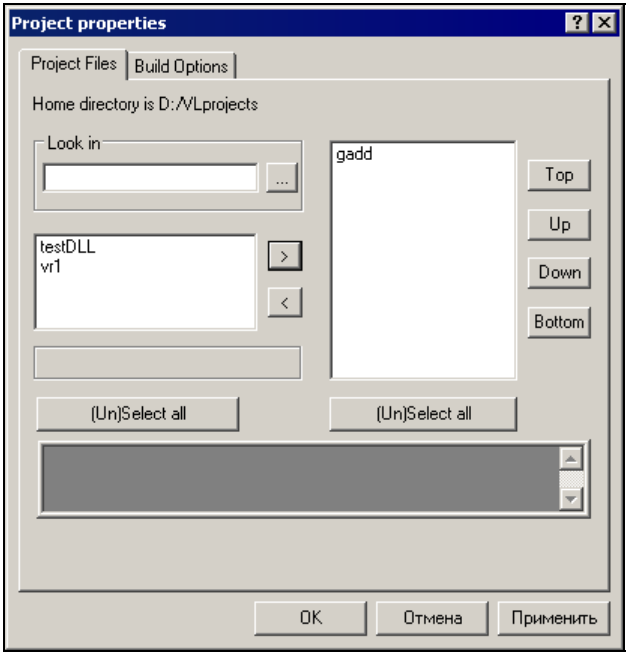


Рис. 2.40. Диалоговое окно **Project properties**, вкладка **Project Files**

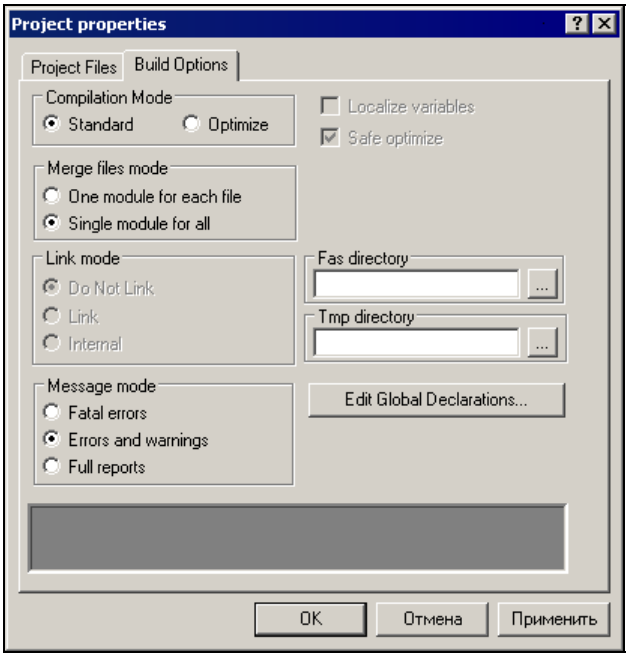


Рис. 2.41. Диалоговое окно **Project properties**, вкладка **Build Options**

Bottom (Нижняя). Чаще всего такое упорядочение не нужно, но иногда оно требуется (например, если есть специальные файлы инициализации переменных, их необходимо вынести наверх, чтобы они загружались первыми).

Вкладка **Build Options** (Параметры сборки) окна **Project properties** (Свойства проекта) (см. рис. 2.41) устанавливает режимы компиляции и связи. В частности, в области **Merge files mode** (Режим слияния файлов) указывается, будут ли FAS-файлы создаваться для каждого LSP-файла отдельно или проект будет компилироваться в единый файл (FAS-приложение). В первом случае должен быть включен переключатель **One module for each file** (Отдельные модули для файлов), во втором — переключатель **Single module for all** (Единый модуль для всех).

На этой же вкладке можно задать папки для размещения FAS-файлов и временных файлов.

Окончание работы в окне завершается щелчком по кнопке **Применить** (Apply). Кнопка **OK** закрывает диалоговое окно, после чего на экране появляется окно открытого проекта. На рис. 2.42 показан пример проекта `MyVL1` из трех файлов.

В верхней части этого окна расположены пять кнопок операций, выполняемых с проектом:

- ♦ — **Project Properties** (Свойства проекта);
- ♦ — **Load Project FAS** (Загрузка FAS-файлов проекта);

- — **Load Source Files** (Загрузка исходных файлов);
- — **Build Project FAS** (Сборка FAS-файла проекта);
- — **Rebuild Project FAS** (Перекомпиляция FAS-файла проекта).

Рис. 2.42. Окно проекта `MyVL1`

Наименования кнопок аналогичны наименованиям пунктов меню **Project** (Проект). В свойства созданных ранее проектов можно вносить изменения, а сами проекты можно загружать как в компилированном виде, так и в виде исходных текстов.

2.4.3. VLX-приложения

Другой вид приложений — это VLX-приложения, в состав которых могут входить не только LSP-файлы, но и DCL-файлы диалоговых окон *(описание процесса их создания см. в главе 3)*, FAS-файлы и текстовые файлы (TXT-файлы). Все включаемые компоненты собираются в единый файл приложения с расширением vlx. Свойства приложения сохраняются в одноименном файле с расширением prv, который используется в случае повторной сборки приложения.

В некоторых предыдущих версиях системы AutoCAD формально в состав VLX-приложения можно было внести DVB-файлы программ на языке VBA (Visual Basic for Applications). Однако к этим программам нельзя было обратиться из среды системы AutoCAD. Поэтому фирма Autodesk исключила DVB-файлы из состава VLX-приложений.

Для создания нового VLX-приложения следует воспользоваться подменю **File | Make Application** (Файл | Собрать приложение) среды Visual LISP. В состав подменю входят следующие пункты:

- **New Application Wizard** (Мастер новых приложений);
- **Existing Application Properties** (Свойства имеющихся приложений);
- **Make Application** (Собрать приложение);
- **Rebuild Application** (Перекомпилировать приложение).

Для создания нового приложения следует выбрать пункт **New Application Wizard** (Мастер новых приложений), который запускает мастер создания новых приложений и открывает страницу **Wizard Mode** (Режим Мастера) (рис. 2.43).

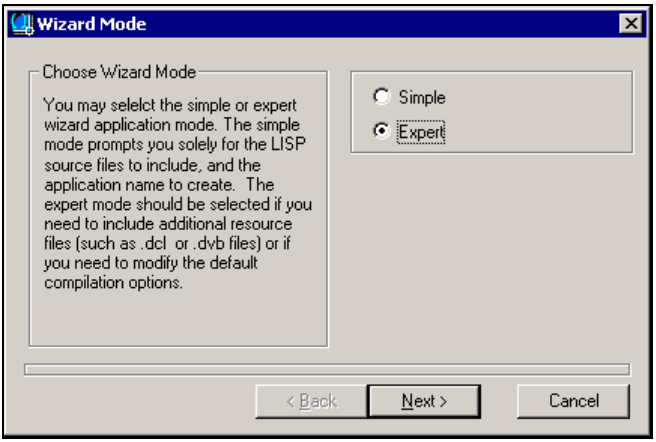


Рис. 2.43. Страница **Wizard Mode**

В этом окне необходимо с помощью соответствующего переключателя задать режим, определяющий тип приложения: **Simple** (Простой) или **Expert** (Сложный). Простое приложение может состоять только из LSP- и FAS-файлов, а сложное — может включать LSP-, FAS-, DCL-, PRJ- и TXT-файлы.

Примечание

Хотя DVB-файлы и исключены из VLX-приложений, однако на странице **Wizard Mode** (Режим Мастера) почему-то осталась фраза **The expert mode should be selected if you need to include additional resource files (such as .dcl or .dvb files)...** (Для включения в приложение дополнительных файлов ресурсов (например, файлов .dcl или .dvb)... следует выбрать сложный режим).

Разберем вариант создания сложного приложения. В случае выбора переключателя **Expert** (Сложный) и щелчка по кнопке **Next** (Далее) открывается страница **Application Directory** (Папка приложения) (рис. 2.44).

Здесь нужно заполнить имя приложения в текстовом поле области **Application Name** (Имя приложения) (например, `My1`), а в текстовом поле области **Application Location** (Расположение приложения) ввести полное имя папки, в которой будут сохранены файл приложения с расширением vlx и файл свойств приложения с расширением prv (основная часть имени этого файла будет взята из имени приложения). Папку удобнее выбирать с помощью кнопки **Browse** (Обзор). После задания имени приложения параметр в поле **Target File** (Целевой файл) покажет имя файла нового приложения с расширением vlx. Щелчком по кнопке **Next** (Далее) переходим к следующей странице — **Application Options** (Параметры приложения) (рис. 2.45).

По умолчанию на данной странице флажок **Separate Namespace** (Отдельное именное пространство) не установлен. Это означает, что функции и глобальные переменные вашего приложения будут доступны другим приложениям.

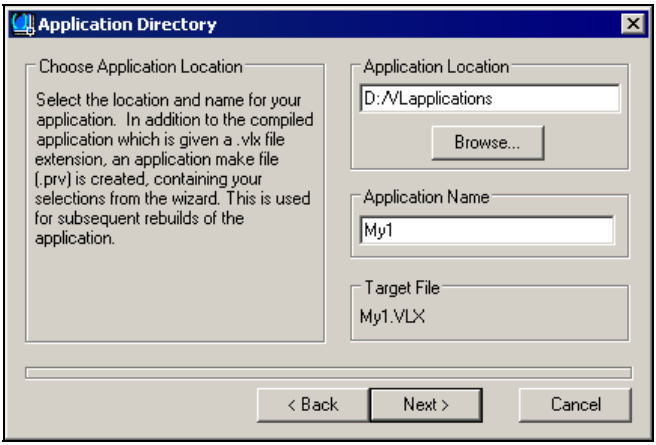


Рис. 2.44. Страница **Application Directory**

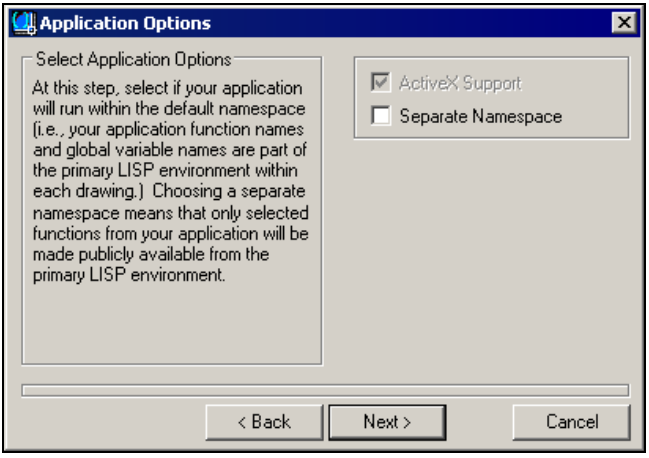


Рис. 2.45. Страница **Application Options**

Если VLX-приложение создано со своим собственным пространством имен, то его функции и глобальные переменные можно будет экспортировать в именное пространство текущего документа с помощью функций `vl-doc-export` и `vl-doc-set`. Функции `vl-doc-import` и `vl-doc-ref` позволяют выполнить импорт функций и переменных в пространство имен приложения.

Замечание

Существует еще внедокументное пространство имен (blackboard). Доступ к его переменным осуществляется применением функций `vl-bb-set` и `vl-bb-ref`.

При установленном флажке **Separate Namespace** (Отдельное именное пространство) пользователь получит доступ к флажку **ActiveX Support** (Поддержка ActiveX) (по умолчанию поддержка включена, но от нее при желании можно отказаться).

Следующая страница (рис. 2.46) — **LISP Files to Include** (LISP-файлы для сборки приложения) — дает возможность перечислить все файлы с расширениями lsp (исходные тексты), fas (компилированные файлы) и prj (файлы проектов), которые необходимо интегрировать в создаваемое приложение. В случае указания PRJ-файла в приложение будут включены все файлы, входящие в этот проект.

Кнопки **Add** (Добавить) и **Remove** (Удалить) позволяют, соответственно, добавлять или удалять компоненты (кнопка **Add** (Добавить) при этом открывает диалоговое окно **Add lisp source files** (Добавление исходных LISP-файлов)). Кнопками **Top** (Начало), **Up** (Вверх), **Down** (Вниз) и **Bottom** (Нижняя) можно регулировать порядок включения файлов в приложение. Раскрывающийся список имеет три элемента (**Lisp source files** (Исходные LISP-файлы), **Compiled Lisp files** (Скомпилированные LISP-файлы) и **Visual LISP project files** (Файлы проектов Visual LISP)), соответствующие типам добавляемых файлов (LSP-, FAS- и PRJ-файлы).

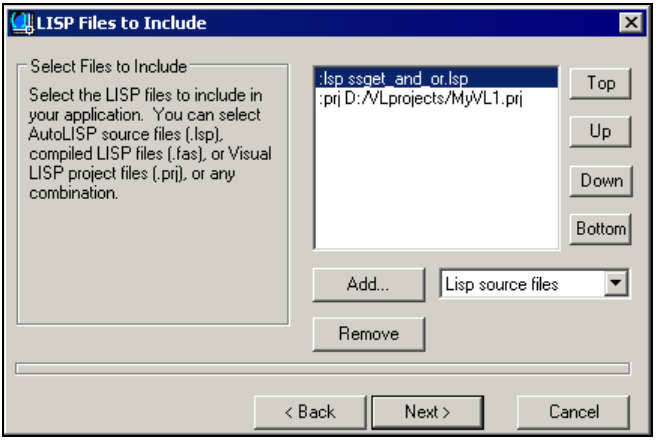


Рис. 2.46. Страница **LISP Files to Include**

Страница **Resource Files to Include** (Файлы ресурсов) (рис. 2.47) не появится, если ранее вы выбрали режим простого приложения (выбран переключатель **Simple** (Простой), см. рис. 2.43).

В качестве дополнительных ресурсов к вашему приложению могут быть добавлены элементы, перечисленные в раскрывающемся списке (три первых типа могли быть выбраны и на предыдущей странице):

- **Lisp source files** (Исходные LISP-файлы);
- **Compiled lisp files** (Скомпилированные LISP-файлы);

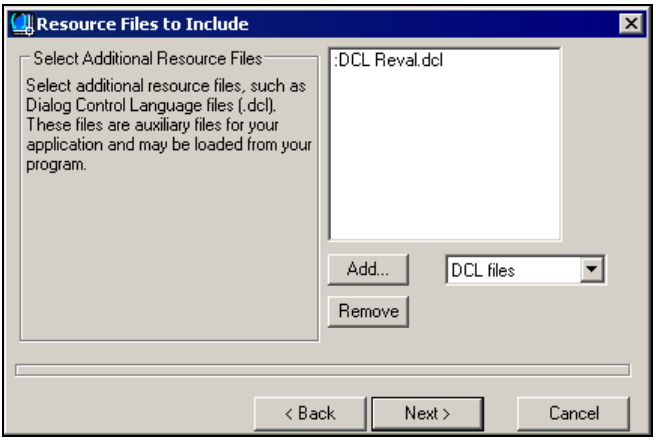


Рис. 2.47. Страница **Resource Files to Include**

- **Visual LISP project files** (Файлы проектов Visual LISP);
- **DCL files** (DCL-файлы);
- **Text files** (Текстовые файлы).

DCL-файлы — это файлы диалоговых окон, структура которых изучается в *главе 3*. *Текстовые файлы* — это любые текстовые файлы с расширением txt, в которых вы храните какую-либо дополнительную информацию. Приложение-хозяин в нужный момент может прочитать эту информацию с помощью функции `vl-get-resource`.

После задания всех дополнительных ресурсов переходим на очередную страницу — **Application Compilation Options** (Параметры компиляции приложения) (рис. 2.48).

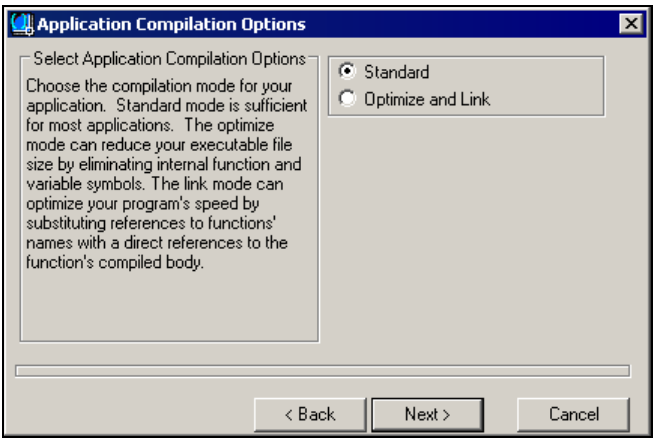


Рис. 2.48. Страница **Application Compilation Options**

В этом окне следует выбрать переключатель, соответствующий нужному режиму компиляции: **Standard** (Стандартная) или **Optimize and Link** (Оптимизация и связывание). Оптимизация уменьшает размер объектного кода и время загрузки приложения, но при неправильном использовании зарезервированных символов может привести к неправильной работе приложения и системы AutoCAD.

Последняя страница мастера создания нового приложения — **Review Selections/Build Application** (Просмотр параметров/Сборка приложения) (рис. 2.49).

Если в данном окне оставить включенным флажок **Build Application** (Собрать приложение) и щелкнуть по кнопке **Finish** (Завершить), Visual LISP создаст файлы приложения (с расширением vlx) и свойств приложения (с расширением prv). Если флажок снять и нажать на кнопку **Finish** (Завершить), то будет создан только PRV-файл, по которому можно в дальнейшем собрать приложение.

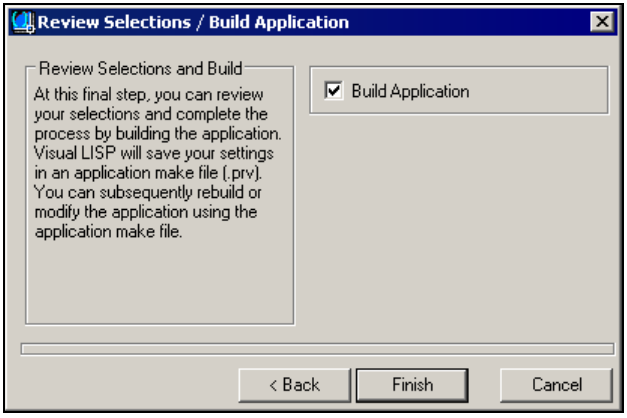


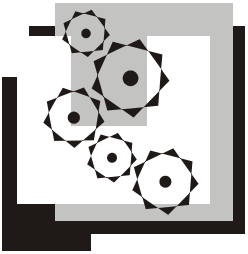
Рис. 2.49. Страница **Review Selections/Build Application**

Для сборки приложения нужно будет воспользоваться пунктом **Make Application** (Собрать приложение) подменю **Make Application** (Собрать приложение) меню **File** (Файл). Пункт **Existing Application Properties** (Свойства имеющихся приложений) того же подменю позволяет просмотреть и изменить свойства или состав приложения, а пункт **Rebuild Application** (Перекомпилировать приложение) — перекомпилировать приложение, если изменились какие-либо его компоненты (DCL-файлы, LSP-файлы и т. д.).

Загрузить в среде Visual LISP собранное приложение можно с консоли с помощью функции LISP `load` или пункта **Load File** (Загрузить файл) меню **File** (Файл).

Если приложение необходимо загрузить в сеансе AutoCAD без среды Visual LISP, то используют либо функцию `load`, либо пункт меню **Tools | AutoLISP | Load** (Сервис | AutoLISP | Загрузить) или пункт **Load Application** (Приложения) того же меню системы AutoCAD.

ГЛАВА 3



Диалоговые окна

Важным компонентом приложений в среде AutoCAD являются диалоговые окна, в которых в нужной форме отображается полезная для пользователя информация и в которых удобно вводить дополнительные данные или настройки, необходимые для правильной и управляемой работы приложения. Приложения, написанные на языке LISP, хранят свои диалоговые окна в текстовых DCL-файлах (файлах с расширением dcl). При разработке VLX-приложений есть возможность не держать DCL-файлы отдельно, а включить их в состав VLX-файлов *(см. главу 2)*.

Примерами DCL-файлов являются файлы acad.dcl и base.dcl, расположенные в папке Support внутри папки Documents and Settings текущего пользователя. В старых версиях системы файлы acad.dcl и base.dcl размещались в папке Support с программным обеспечением системы AutoCAD.

Особую роль играет файл base.dcl, который содержит определения базовых элементов (областей, полей, кнопок, списков, флажков, переключателей и т. д.).

В данной главе на примерах показано, как работать с DCL-окнами с использованием LISP-программ.

Более сложные окна, с использованием инструментов Windows API и MFC, можно создавать, применяя Microsoft Visual Studio и специальный пакет ObjectARX, предназначенный для разработки на языках C++, C# и Visual Basic.NET приложений для системы AutoCAD. В книге "AutoCAD: разработка приложений, настройка и адаптация"[1] приведены примеры работы с ObjectARX, а также описание еще одного пакета — ObjectDCL фирмы 3rd day Software (Канада), который расширяет количество типов диалоговых окон, доступных для LISP-приложений.

---

[1] Полещук Н. Н. AutoCAD: разработка приложений, настройка и адаптация. — СПб.: БХВ-Петербург, 2006. — 992 с. (+CD-ROM).

3.1. Язык DCL

Для диалоговых окон разработан специальный язык DCL (Dialog Control Language), синтаксис которого имеет много общего с языком С.

3.1.1. Директивы

Файл, написанный на языке DCL, состоит из логических единиц — *директив*, которые записываются в свободном формате.

Директивы располагаются в файле последовательно. Каждая директива описывает элемент (tile) или способ группировки элементов при выравнивании внутри диалогового окна. Есть также директива для описания *диалога* (под диалогом мы понимаем совокупность директив, отображаемую в виде одного диалогового окна). В одном файле могут быть записаны один или несколько диалогов. Перед тем как открыть диалоговое окно, система должна выполнить загрузку DCL-файла, содержащего описание диалога, соответствующего открываемому окну, и вызвать нужный диалог на экран.

Не обязательно начинать директивы с новой строки (хотя опыт показывает, что, начиная каждую директиву с новой строки, вы облегчаете себе последующее чтение файла). Признаком *конца директивы* (и, соответственно, разделителем между директивами) является точка с запятой или закрывающая фигурная скобка. Скобки должны быть сбалансированы (количество открывающих скобок должно быть равно количеству закрывающих). Между *ключевыми словами* языка, *разделительными символами* (к которым относятся двоеточие, точка с запятой, знак равенства, фигурные скобки), *метками*, числовыми и строковыми значениями может быть любое количество пробелов. Текстовые строки должны заключаться в двойные кавычки.

Все элементы языка DCL являются чувствительными к регистру, поэтому нельзя, например, вводить ключевое слово в верхнем регистре, если оно дальше используется в нижнем. Все ключевые слова, метки и *коды элементов* должны состоять только из латинских букв (верхнего и нижнего регистров), цифр и символа подчеркивания и обязательно начинаться с буквы.

Директивы могут записываться в одном из двух видов. Первый вид директивы:

```
[<метка>]:<оператор>{<атрибуты>}
```

Здесь квадратные скобки обозначают необязательность стоящего внутри них элемента. Фигурные скобки и двоеточие являются обычными символами.

Например:

```
Edb1:edit_box{label="Направление:";edit_width=12;value="";key="Dru";}
```

Второй вид директивы:

```
<метка>;
```

Пример такой директивы:

```
ok_cancel_help_info;
```

Параметр <метка> — метка директивы (по этой метке можно ссылаться на данную директиву в других местах текста описания диалогового окна). Метка — идентификатор директивы, который должен быть уникальным в данном файле. В примере директивы первого типа метка директивы — Edb1. В примере директивы второго типа метка ok_cancel_help_info является ссылочной меткой на группу кнопок, описанную в базовом файле base.dcl системы AutoCAD.

Параметр <оператор> — либо ключевое слово определения диалога, либо ключевое слово с типом элемента, либо метка другой директивы с типом элемента, либо ключевое слово группирования элементов.

Замечание

В параметре <оператор> может быть указано сразу несколько меток, разделяемых двоеточиями. В этом случае система AutoCAD берет значения атрибутов из элемента с первой меткой, дополняет их значениями других атрибутов из элемента со второй меткой и т. д. Пример такой директивы:

```
mn:oper1:oper2{label="Double";}
```

Здесь mn — метка новой директивы, а oper1 и oper2 — метки ранее созданных директив. Если имена атрибутов, использованных в директивах oper1 и oper2, повторяются, приоритет имеет первая директива.

В приведенном выше примере директивы первого типа в качестве параметра <оператор> используется ключевое слово edit_box, применяемое для редактируемых текстовых полей.

Если директива описывает элемент диалогового окна, то параметр <атрибуты> — это описание атрибутов элемента и их значений (записываются только те атрибуты, значения которых отличны от значений по умолчанию).

Параметр <атрибуты> — это последовательность разделяемых точкой с запятой операций задания атрибутам значений, которая имеет вид:

```
<атрибут1>=<значение1>;[<атрибут2>=<значение2>;
[ ... <атрибутN>=<значениеN>]...]
```

Наименования атрибутов должны быть стандартными ключевыми словами. Значения атрибутов могут быть числами или строками (тип значения зависит от атрибута).

Если директива является директивой группировки элементов (т. е. объединения в колонку или ряд), то она содержит внутри фигурных скобок директивы определения группируемых элементов.

3.1.2. Пример диалогового окна

На рис. 3.1 приведен пример диалогового окна пользовательского приложения.

В листинге 3.1 приведен текст файла d_tpos.dcl, соответствующего диалоговому окну, изображенному на рис. 3.1.

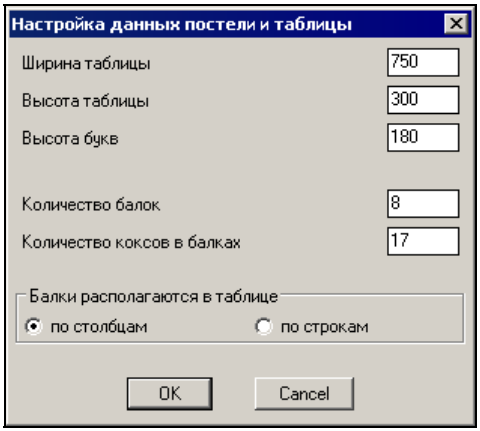


Рис. 3.1. Пример диалогового окна

Листинг 3.1. Файл d_tpos.dcl

```
// Глава 03\Book01\d_tpos.dcl
np_tpos: dialog{label="Настройка данных постели и таблицы";
  :edit_box{label="Ширина таблицы";key="kShir";value="750";edit_width=6;}
  :edit_box{label="Высота таблицы";key="kVys";value="300";edit_width=6;}
  :edit_box{label="Высота букв";key="kHbuk";value="180";edit_width=6;}
  :spacer{height=1;}
  :edit_box{label="Количество балок";key="kNumbalk";
            value="8";edit_width=6;}
  :edit_box{label="Количество коксов в балках";key="kNumkoks";
            value="17";edit_width=6;}
  :spacer{height=1;}
  :radio_row{label="Балки располагаются в таблице";
    :radio_button{label="по столбцам";key="kPoStolb";value="1";}
    :radio_button{label="по строкам";key="kPoStrok";value="0";}
  }
  :spacer{height=1;}
  ok_cancel;
}// конец np_tpos
```

В этом листинге имеется 17 строк. В целом они образуют директиву описания диалога, которому присвоено имя `np_tpos` (это имя записано в метке в начале

первой строки). Открывающая фигурная скобка этой директивы находится на первой строке, закрывающая — на последней. Подробнее о директиве описания диалога *см. следующий раздел*.

Директивы, расположенные в строках со второй по шестнадцатую, не только являются элементами директивы описания диалога (поскольку расположены внутри фигурных скобок), но также являются директивами, описывающими элементы стандартных типов `edit_box` (редактируемое текстовое поле), `spacer` (промежуток), `radio_row` (область с рядом переключателей) и `ok_cancel` (ряд из кнопок **OK** и **Cancel** (Отмена)). Подробнее *см. разд. 3.3*.

Далее в листинге 3.2 показано, как вызвать данное диалоговое окно из программы на языке LISP.

В начале DCL-файла могут быть строки с инструкциями копирования в данный файл содержимого других DCL-файлов, например:

```
@include "fourth-b.dcl"
```

Данная инструкция при загрузке основного DCL-файла копирует в оперативную память содержимое файла fourth-b.dcl (если имя файла указано без пути, то нужно, чтобы система AutoCAD могла обнаружить этот файл с помощью путей поиска файлов поддержки).

Не допускается с помощью инструкции `@include` загружать файл acad.dcl со стандартными диалогами системы AutoCAD.

После инструкций `@include` в файле могут располагаться директивы описания элементов, которые в явном виде не входят ни в какой диалог, но с помощью меток ссылки на них могут быть добавлены в следующие далее диалоги.

Если в тексте файла встречаются символы //, то для системы AutoCAD это означает, что далее до конца строки идет *комментарий*.

Написание и отладка DCL-файлов значительно облегчаются при использовании текстового редактора среды Visual LISP *(см. главу 2)*.

3.1.3. Директива описания диалога

Директива описания диалога задает имя, по которому диалоговое окно из программы вызывается на экран. Если в файле находится описание нескольких диалогов, они могут следовать в любом порядке, независимо от последовательности их фактического вызова приложением системы AutoCAD.

Директива описания диалога использует ключевое слово `dialog` и имеет следующую форму:

```
<имя>:dialog{<атрибуты> <элементы>}
```

В качестве параметра `<имя>` может использоваться любой идентификатор (метка), который и задает имя диалога. Имена диалогов в одном сеансе AutoCAD не должны повторяться, иначе может произойти замена одного диалога другим.

Параметр `<атрибуты>` представляет собой последовательность атрибутов и их значений следующего вида: `<атрибут> = <значение>;`.

Параметр `<элементы>` является набором директив описания элементов (т. е. кнопок, списков, слайдов, надписей и других элементов).

Атрибуты диалога — это различные характеристики окна в целом или его элементов. В табл. 3.1 приведены ключевые слова, которые могут использоваться в качестве атрибутов в директиве описания диалога.

Таблица 3.1. *Атрибуты директивы описания диалога*

Атрибут	**Описание**	**Допустимые значения**	**Значение по умолчанию**
`label`	Начальный текст заголовка диалогового окна	Строка	`" "` (пробел)
`value`	Текущий текст заголовка диалогового окна	Строка	Нет
`key`	Код диалога	Строка	Нет
`width`	Ширина окна	Число	Нет
`height`	Высота окна	Число	Нет
`initial_focus`	Код подсвеченного элемента в диалоговом окне	Строка	Нет
`children_alignment`	Способ выравнивания подчиненных элементов по умолчанию	`left`, `right`, `centered` (для колонок); `top`, `bottom`, `centered` (для рядов)	`left` (для колонок); `centered` (для рядов)
`children_fixed_width`	Фиксация ширины подчиненных элементов по умолчанию	`true`, `false`	`false`
`children_fixed_hight`	Фиксация высоты подчиненных элементов по умолчанию	`true`, `false`	`false`

Если в описании диалога использованы лишние атрибуты, они игнорируются.

Атрибуты `label` и `value` близки по смыслу и в какой-то степени дублируют друг друга. Различие между ними заключается в следующем.

Прочитав при загрузке DCL-файла описание диалога и входящих в него элементов, система AutoCAD по своим правилам определяет форму и размер выводимого диалогового окна. Вычисляются минимальные размеры каждого элемента, учитывается последовательность расположения элементов. Каждый элемент до-

полняется окружающим его свободным пространством и в результате определяется размер прямоугольника, в который выводится диалоговое окно. Строковое значение, которое задано в атрибуте `label`, участвует в определении размера заголовка диалогового окна и влияет таким образом на размер всего окна. Значение атрибута `value` уже не влияет на размер окна.

Если задано значение атрибута `value`, в заголовок выводится оно, а не значение атрибута `label`. При этом, если значение атрибута `value` короче, чем значение атрибута `label`, в конец заголовка добавляются пробелы. Если значение атрибута `value` оказалось длиннее, чем значение атрибута `label`, и не хватает места для его вывода, тогда размеры диалогового окна не увеличиваются, а текст, выводимый в заголовок, обрывается с помощью многоточия.

Можно программно изменить заголовок выводимого диалогового окна с помощью функции `set_tile`. Для этой цели используется атрибут `key`, называемый *кодом диалога* (ключевое слово `key` для него сохранено как для элементов). В качестве значения кода диалога могут использоваться только строки.

Атрибут `width` задает минимальную ширину диалогового окна, меньше которой она не может быть сформирована системой AutoCAD при выводе на экран. Ширина диалогового окна вначале вычисляется системой по размерам заголовка и элементов, а затем увеличивается, если она получилась меньше, чем значение атрибута `width`. Значение атрибута задается количеством условных символов, ширина которых примерно равна средней ширине символов верхнего и нижнего регистра латинского алфавита для шрифта, действующего по умолчанию.

Атрибут `height` по своему смыслу аналогичен атрибуту `width`, но задает минимальную высоту диалогового окна количеством условных символов с высотой, равной максимальной высоте букв действующего шрифта.

Атрибут `initial_focus` указывает код элемента, который подсвечивается по умолчанию и выполняет действие при нажатии пользователем клавиши <Enter>.

Три атрибута с префиксом `children_` задают свойства не самого диалога, а подчиненных (входящих в него) элементов.

Атрибут `children_alignment` задает способ, которым система AutoCAD выравнивает элементы, входящие в диалоговое окно. Если элементы диалога расположены в колонку (вертикально), в качестве значения атрибута `children_alignment` могут использоваться такие значения, как:

- `left` — выравнивание влево (справа добавляется свободное пространство);
- `right` — выравнивание вправо (слева добавляется свободное пространство);
- `centered` — выравнивание по центру (слева и справа добавляется свободное пространство).

В качестве значения, действующего по умолчанию (т. е. когда атрибут `children_alignment` не задан), используется `left`.

Если элементы диалога расположены в ряд (горизонтально), могут использоваться следующие значения атрибута:

- `top` — выравнивание вверх (снизу добавляется свободное пространство);
- `bottom` — выравнивание вниз (сверху добавляется свободное пространство);
- `centered` — выравнивание посередине (снизу и сверху добавляется свободное пространство).

По умолчанию в этом случае действует значение `centered`.

На элементы, в которых задан свой атрибут выравнивания (`alignment`), значение атрибута `children_alignment` не влияет.

Атрибуты `children_fixed_width` и `children_fixed_height` обычно задают, если они имеют значение `true`. Это означает, что по умолчанию размеры подчиненных элементов (например, кнопок) при выводе диалогового окна не увеличиваются, соответственно, по ширине и высоте (система не может растягивать кнопки, а может только увеличивать расстояние между ними).

Если вы при написании диалоговых окон пользуетесь средой Visual LISP, то в вашем распоряжении есть способ проверки внешнего вида диалогового окна без загрузки DCL-файла в память. Для этого вы можете воспользоваться подменю **Tools | Interface Tools** (Сервис | Инструменты интерфейса) среды Visual LISP *(см. главу 2)*, которое имеет два пункта:

- **Preview DCL in Selection** (Просмотр DCL из фрагмента);
- **Preview DCL in Editor** (Просмотр DCL из редактора).

Первый пункт используется, когда в редактируемом DCL-файле выделен фрагмент, содержащий полное описание одного диалога, а второй — когда выделенного фрагмента нет и проверке подвергается весь текст файла.

После выбора одного из этих подпунктов система AutoCAD выводит окно **Enter the dialog name** (Введите имя диалогового окна), приведенное на рис. 3.2.

Рис. 3.2. Окно **Enter the dialog name**

В центральной части окна нужно ввести или выбрать из раскрывающегося списка имя проверяемого диалога. После этого система AutoCAD отобразит проверяемое диалоговое окно или выведет сообщение об ошибке.

3.2. Открытие и закрытие диалоговых окон

Для того чтобы вызвать на экран диалоговое окно, необходимо сначала загрузить DCL-файл, содержащий нужный диалог, а затем уже открыть диалоговое окно.

В данном разделе рассматриваются LISP-функции, которые позволяют загружать и выгружать DCL-файлы (`load_dialog`, `unload_dialog`), открывать и закрывать окна загруженных диалогов.

3.2.1. Функция *load_dialog*

Функция `load_dialog` загружает DCL-файл в память. Синтаксис функции:

```
(load_dialog <файл>)
```

Аргумент `<файл>` — текстовая строка с именем загружаемого DCL-файла.

Возвращаемое значение — целое число (отрицательное, если загрузка не выполнилась; положительное, если внутренний номер загруженного файла в памяти).

Если имя файла в аргументе указано без полного пути, то система AutoCAD пытается найти файл по стандартному алгоритму поиска файлов поддержки.

Пример:

- `(setq dcl_id (load_dialog "np_tpos.dcl"))` может вернуть –1 в случае ошибки (файл не обнаружен) или положительное число (порядковый номер DCL-файла, загруженного в данном сеансе работы с системой AutoCAD) — например, 5.

Для выгрузки DCL-файла из памяти следует использовать функцию `unload_dialog`.

3.2.2. Функция *unload_dialog*

Функция `unload_dialog` выгружает DCL-файл из памяти. Синтаксис функции:

```
(unload_dialog <номер>)
```

Аргумент `<номер>` — положительное целое число с номером загруженного DCL-файла (номер был возвращен предшествующим вызовом функции `load_dialog`).

Возвращаемое значение — `nil`.

DCL-файлы после закрытия описанных в них диалоговых окон должны выгружаться, освобождая память.

Пример (продолжение примера из предыдущего подраздела):

- `(unload_dialog dcl_id)` возвращает `nil`.

3.2.3. Функция *new_dialog*

Функция `new_dialog` открывает на экране диалоговое окно, если его диалог уже загружен в память в составе DCL-файла.

```
(new_dialog <диалог> <номер> [<действие> [<положение>]])
```

Аргументы: `<диалог>` — текстовая строка с именем диалога; `<номер>` — положительное целое число с номером загруженного DCL-файла данного диалога; `<действие>` — строка с LISP-выражением, выполняемым по умолчанию (если никакого LISP-выражения выполнять не надо, то следует в качестве значения задать пустую строку — `""`); `<положение>` — список из двух чисел, задающий в пикселах положение левого верхнего угла диалогового окна (левый верхний угол экрана имеет координаты (0, 0); если задать значение `'(-1 -1)`, то окно расположится в центре экрана, что является положением окна по умолчанию).

Возвращается `T`, если диалоговое окно открылось, и `nil`, если возникла ошибка открытия.

Если по условиям работы приложения аргумент `<положение>` должен быть задан, тогда обязательно должен быть задан и аргумент `<действие>`. Система AutoCAD выполняет LISP-выражение, заданное в аргументе `<действие>`, если пользователь выбрал в диалоговом окне элемент, для которого не задано действие (с помощью функции `action_tile` в LISP-программе или с использованием атрибута `action` в DCL-файле). О функции `action_tile` *см. разд. 3.4.1*, об атрибуте `action` — *разд. 3.3.3*.

Пример:

- `(new_dialog "np_tsp" dcl_id "" (list 222 114))` открывает диалоговое окно с диалогом `np_tsp`, располагая его левый верхний угол в точке с координатами (222, 114), и возвращает `T`.

3.2.4. Функция *start_dialog*

Функция `start_dialog` активизирует диалоговое окно и готовит систему AutoCAD к приему действий пользователя в этом окне. Синтаксис:

```
(start_dialog)
```

Возвращает целое число, которое было передано в качестве аргумента функции `done_dialog` (эта функция закрывает диалоговое окно — *см. разд. 3.2.5*). Если приложение закрывает диалоговое окно, а функции `done_dialog` при этом не передано никакого аргумента, то функция `start_dialog` возвратит: `1`, если пользователь выйдет из диалогового окна с помощью кнопки **OK**; `0`, если пользователь выйдет из окна с помощью кнопки **Cancel** (Отмена). Кроме того, функция `start_dialog` возвратит `-1`, если работа всех диалоговых окон будет прервана с помощью функции `term_dialog`.

3.2.5. Функция *done_dialog*

Функция done_dialog закрывает диалоговое окно и передает значение, которое будет возвращаемым значением функции start_dialog. Синтаксис:

```
(done_dialog [<число>])
```

Аргумент <число> — это положительное целое число, которое станет возвращаемым значением функции start_dialog.

Возвращаемое значение — список из двух целых чисел с координатами левого верхнего угла диалогового окна в момент его закрытия.

Пример:

- (done_dialog 92) может, например, вернуть список (229 160), а функция start_dialog при этом вернет 92.

Замечание

Функция done_dialog используется только внутри LISP-выражений в функции action_tile или атрибуте action.

3.2.6. Функция *term_dialog*

Функция term_dialog закрывает все открытые диалоговые окна, как будто пользователь вышел из каждого из них с помощью кнопки **Cancel** (Отмена). Синтаксис:

```
(term_dialog)
```

Возвращаемое значение — nil.

Если работа приложения прерывается по какой-то причине, а некоторые диалоговые окна остались не закрытыми, то система AutoCAD сама обеспечивает вызов функции term_dialog.

3.2.7. Пример вызова диалогового окна

Рассмотрим простой пример пользовательской функции, которая открывает диалоговое окно, изображенное на рис. 3.1, и закрывает его, когда пользователь нажимает кнопку **OK** или **Cancel** (Отменить). Текст программы приведен в листинге 3.2.

Листинг 3.2. Пример функции show_tpos, работающей с диалоговым окном

```
; Глава 03\Book01\show_tpos.lsp
(defun show_tpos ( / dcl_id)
  (if (< (setq dcl_id (load_dialog "d_tpos.dcl")) 0)(exit))
```

```
  (if (not (new_dialog "np_tpos" dcl_id)) (exit))
  (start_dialog)
  (unload_dialog dcl_id)
  (princ)
);defun show_tpos
```

Эта функция работает с диалогом из DCL-файла d_tpos.dcl, текст которого дан в листинге 3.1. Файл d_tpos.dcl должен находиться в одном из путей, по которому система AutoCAD выполняет поиск вспомогательных файлов, или же в имени файла следует указать полный путь (например, `"C:\\Temp\\d_tpos.dcl"`).

Функция имеет одну рабочую переменную `dcl_id`, в которой сохраняется внутренний номер открываемого диалогового окна.

Во второй строке программы выполняется загрузка файла d_tpos.dcl, и если загрузка не выполнилась (т. е. переменная `dcl_id` получила отрицательное значение, возвращенное функцией `load_dialog`), то выполняется функция `exit`, прерывающая работу нашего приложения.

В третьей строке проверяется, имеется ли в загруженном DCL-файле диалог с именем `np_tpos`. Если нет, то снова работа программы завершается с помощью функции `exit`.

В четвертой строке функция `start_dialog` открывает диалоговое окно и ждет нажатия пользователем кнопки **OK** или **Cancel** (Отменить).

В программе нет выражений с функцией `done_dialog`, которая является единственным инструментом закрытия диалогового окна. Однако поскольку в описании использованы кнопки **OK** или **Cancel** (Отменить), то это и обеспечивает неявное применение функции `done_dialog`. В программу должен быть добавлен разбор введенных значений полей и нажатых кнопок.

В конце программы функция `unload_dialog` выгружает DCL-файл из памяти, а функция `princ` обеспечивает тихий выход (т. е. без возвращаемого значения).

3.3. Элементы диалога

Внутри директивы описания диалога находятся директивы описания элементов. Каждый элемент является либо *элементом оформления* (надписью, слайдом и т. п.), либо *элементом управления* (кнопкой, списком и т. п., к которым пользователь имеет доступ с помощью клавиатуры или устройства указания).

3.3.1. Типы элементов

В директивах в качестве значения условного элемента `<оператор>` могут фигурировать следующие наименования элементов:

- `button` — кнопка;
- `edit_box` — редактируемое текстовое поле;

- `errtile` — поле для отображения сообщений об ошибках;
- `image` — поле с изображением;
- `image_button` — кнопка с изображением;
- `list_box` — список;
- `popup_list` — раскрывающийся список;
- `radio_button` — переключатель (элемент группы переключателей);
- `radio_column` — вертикальная группа (колонка) переключателей;
- `radio_row` — горизонтальная группа (ряд) переключателей;
- `boxed_radio_column` — колонка переключателей в рамке;
- `boxed_radio_row` — ряд переключателей в рамке;
- `slider` — скользящая шкала;
- `toggle` — флажок;
- `column` — колонка (средство объединения элементов);
- `row` — ряд (средство объединения элементов);
- `text` — текстовое нередактируемое поле;
- `text_part` — часть текста, включаемая в элементы `concatenation` и `paragraph`;
- `concatenation` — горизонтальное соединение элементов `text_part`;
- `paragraph` — вертикальное соединение элементов `text_part` и `concatenation`;
- `spacer` — пустой элемент произвольной ширины и высоты (нужно обязательно задать атрибуты `width` и `height`);
- `spacer_0` — элемент `spacer` с нулевыми значениями атрибутов `width` и `height`;
- `spacer_1` — элемент `spacer` с единичными значениями атрибутов `width` и `height`.

Все эти элементы описаны в базовом файле base.dcl.

3.3.2. Элементы с кнопкой выхода

В файле base.dcl определены также следующие элементы, ссылки на которые очень часто применяются в пользовательских диалоговых окнах:

- `ok_button` — элемент с одной кнопкой **OK**;
- `ok_only` — элемент в виде колонки из одного элемента с кнопкой **OK**;
- `ok_cancel` — объединенный элемент из кнопок **OK** и **Cancel** (Отмена);
- `ok_cancel_help` — объединенный элемент из кнопок **OK**, **Cancel** (Отмена) и **Help** (Справка);

- `ok_cancel_err` — объединенный элемент из кнопок **OK** и **Cancel** (Отмена) и расположенного строкой ниже поля сообщений об ошибках;
- `ok_cancel_help_errtile` — объединенный ряд из кнопок **OK**, **Cancel** (Отмена) и **Help** (Справка) и расположенное строкой ниже поле сообщений об ошибках;
- `ok_cancel_help_info` — объединенный ряд из кнопок **OK**, **Cancel** (Отмена), **Help** (Справка) и **Info** (Инфо).

Все эти элементы обладают общим свойством — содержат кнопку выхода из диалогового окна (нажатие на нее закрывает окно).

Замечание

Диалог должен обязательно содержать хотя бы один элемент с кнопкой выхода — иначе диалоговое окно будет невозможно закрыть. Таким элементом может быть как один из только что перечисленных стандартных элементов, так и любой другой элемент, в функции действия которого имеется обращение к функции `done_dialog`.

В табл. 3.2 приведены коды (keys) стандартных элементов, включенные в элементы с кнопкой выхода.

Таблица 3.2. *Коды стандартных элементов*

Кнопка (элемент)	Код элемента
OK	`"accept"`
Cancel (Отмена)	`"cancel"`
Help (Справка)	`"help"`
Info (Инфо)	`"info"`
Поле сообщений об ошибках	`"error"`

3.3.3. Атрибуты элементов

В табл. 3.3 приводится алфавитный перечень всех атрибутов, встречающихся в директивах описания и группировки элементов.

Таблица 3.3. *Атрибуты директив описания и группировки элементов*

Атрибут	Описание	Допустимые значения	Область применения	Значение по умолчанию
`action`	Действия, выполняемые при выборе данного элемента	Строка, содержащая текст выражений языка LISP	Элементы управления	`" "`

Таблица 3.3 *(продолжение)*

Атрибут	Описание	Допустимые значения	Область применения	Значение по умолчанию
alignment	Способ выравнивания	left, right, centered; top, bottom, centered	Все элементы	left, centered
allow_accept	Признак активизации элемента при нажатии клавиши <Enter>	true, false	Элементы управления	false
aspect_ratio	Коэффициент отношения ширины элемента к высоте	Вещественное число (если 0.0, то коэффициент принимается равным коэффициенту пропорциональности слайда)	image	Нет
big_increment	Большой шаг скользящей шкалы	Целое число	slider	0.1 от разности между max_value и min_value
children_ alignment	Задание через родительский элемент способа выравнивания сгруппированных элементов	left, right, centered (колонка); top, bottom, centered (ряд)	row, column, boxed_row, boxed_column	left (колонка); centered (ряд)
children_ fixed_height	Задание через родительский элемент фиксации высоты сгруппированных элементов	true, false	row, column, boxed_row, boxed_column	false
children_ fixed_width	Задание через родительский элемент фиксации ширины сгруппированных элементов	true, false	row, column, boxed_row, boxed_column	false
color	Цвет	Числа 0-256 или dialog_line, dialog_foreground, dialog_background, graphics_background, black, red, yellow, green, cyan, blue, magenta, whitegraphics_ foreground	image	7
edit_limit	Предельный размер редактируемого поля (количество символов)	Целое число (не более 256)	edit_box	132

Таблица 3.3 *(продолжение)*

Атрибут	Описание	Допустимые значения	Область применения	Значение по умолчанию
edit_width	Размер видимой части редактируемого поля (количество символов)	Целое число	edit_box	Максимально возможное в данном окне
fixed_height	Фиксация высоты элемента	true, false	Элементы управления	false
fixed_width	Фиксация ширины элемента	true, false	Элементы управления	false
fixed_width_ font	Фиксация ширины символов шрифта	true, false	list_box, popup_list, text, text_part	false
height	Минимальная высота элемента	Число	Все элементы	Нет
initial_focus	Код элемента, выбранного по умолчанию	Строка	Элементы управления	Нет
is_bold	Признак выбора полужирного шрифта	true, false	Все элементы	false
is_cancel	Признак срабатывания при нажатии клавиши <Esc>	true, false	button (только один элемент в диалоге может иметь значение true)	false
is_default	Признак срабатывания при нажатии клавиши <Enter>	true, false	button (только один элемент в диалоге может иметь значение true)	false
is_enabled	Признак того, что элемент не погашен в начальном состоянии окна	true, false	Элементы управления	true
is_tab_stop	Признак возможности выбора элемента с помощью клавиши <Tab>	true, false	Элементы управления	true
key	Код элемента	Строка	Все элементы	Нет
label	Начальный текст элемента	Строка	Все элементы	" "
layout	Ориентация скользящей шкалы	horizontal, vertical	slider	horizontal

Таблица 3.3 *(окончание)*

Атрибут	Описание	Допустимые значения	Область применения	Значение по умолчанию
list	Начальное состояние списка	Строка с элементами списка (разделителем элементов является \n)	list_box, popup_list	Нет
max_value	Верхнее значение шкалы	Целое число (не более 32 767)	slider	10 000
min_value	Нижнее значение шкалы	Целое число (не менее –32 768 и не более max_value)	slider	0
mnemonic	Символ быстрого выбора элемента	Буква	Элементы управления	Нет
multiple_select	Признак разрешения множественного выбора	true, false	list_box	false
password_char	Символ, закрывающий текст вводимого пароля	Символ	edit_box	Нет
small_increment	Малый шаг скользящей шкалы	Целое число	slider	0.01 от разности между max_value и min_value
tabs	Задание номеров позиций для табуляции	Строка с числами (например, "8 16 24 32")	list_box, popup_list	Нет
tab_truncate	Признак обрезки длинных значений при переходе за табуляцию	true, false	list_box, popup_list	false
value	Начальное значение	Строка	Все элементы	Нет
width	Минимальная ширина элемента	Число	Все элементы	Нет

Из таблицы видно, что большая часть атрибутов (например, label, value, key, width, height и т. д.) применяется во всех элементах. Однако необходимо учитывать, что смысл атрибута от элемента к элементу может несколько меняться.

Есть специальные атрибуты, употребляемые только в одном или двух типах элементах (например, layout, multiple_select, password_char, big_increment, small_increment, tabs, tab_truncate и т. д.).

При описании элемента нет необходимости перечислять значения всех его атрибутов. Достаточно указать только те атрибуты, для которых вы хотели бы использовать значения, отличные от значений по умолчанию.

3.3.4. Примеры элементов

В этом разделе приводятся примеры описания основных элементов диалога.

3.3.4.1. Кнопка

Элемент `button` является самым распространенным в диалоговых окнах. Он имитирует кнопку, которую нужно нажать пользователю для выполнения определенного действия. В листинге 3.3 приведен пример использования полей типа `button`.

Листинг 3.3. Пример диалога с элементом `button`

```
// Глава 03\Book02\sample1.dcl
sample1: dialog {label="Пример диалога с кнопками";
:button{label="Кнопка 1"; key="b1"; action="(setq bstyle 1)";}
:button{label="Кнопка 2"; fixed_width=true; alignment=right; key="b2";
        action="(setq bstyle 2)";}
ok_button;
}// конец sample1
```

На рис. 3.3 представлено соответствующее диалоговое окно. Обратите внимание, что горизонтальный размер первой кнопки выбран системой AutoCAD на полную ширину окна, а у второй кнопки, для которой задано значение `true` атрибута `fixed_width`, размер сформирован минимальным (таким, чтобы только разместилась надпись). Кроме того, с помощью значения `right` атрибута `alignment` вторая кнопка прижата вправо.

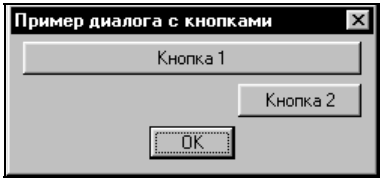


Рис. 3.3. Пример использования элемента `button`

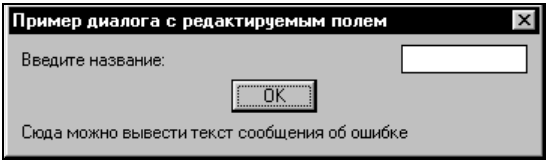


Рис. 3.4. Пример использования элементов `edit_box` и `errtile`

Ссылка `ok_button` размещает в диалоговом окне стандартную кнопку выхода.

Атрибут `action` задает похожие LISP-выражения, которые выполняются при нажатии обеих кнопок (при нажатии первой кнопки переменной `bstyle` присваивается значение 1, при нажатии второй кнопки — значение 2). Действия, выполняемое системой AutoCAD при нажатии кнопок, можно было бы вместо атрибута `action` задать с помощью функции `action_tile` в программе, которая открывает данное диалоговое окно.

3.3.4.2. Текстовые поля

В листинге 3.4 приведен пример использования полей двух типов: `edit_box` и `errtile`.

Листинг 3.4. Пример диалога с элементами `edit_box` и `errtile`

```
// Глава 03\Book02\sample2.dcl
sample2: dialog {label="Пример диалога с редактируемым полем";
:edit_box{label="Введите название:"; key="b1";
          edit_width=12; edit_limit=24;}
ok_button;
:errtile{label="Сюда можно вывести текст сообщения об ошибке";}
}// конец sample2
```

На рис. 3.4 показано формируемое диалоговое окно. Длина видимой части редактируемого поля, в котором пользователь может вводить текст, с помощью атрибута `edit_width` задана равной 12 символам. Предельная длина вводимого текста (см. задание атрибута `edit_limit`) — 24 символа. Избавиться от слишком широкого промежутка между заголовком поля `edit_box` и областью ввода текста можно либо с помощью значения `true` атрибута `fixed_width`, либо путем уменьшения длины текста в заголовке диалогового окна (тогда система добавит меньше пустого места).

Внутри текстового поля `errtile`, предназначенного для размещения информации об ошибках, есть элемент с кодом `"error"`. В этот элемент с помощью функции `set_tile` можно из программы выводить необходимые сообщения для пользователя, работающего с диалоговым окном *(см. разд. 3.4)*.

3.3.4.3. Элементы с изображениями

В листинге 3.5 приведен пример использования элементов `image` и `image_button`. Первый из них является чисто оформительским, а второй — кнопкой с графическим изображением.

Листинг 3.5. Пример диалога с элементами `image` и `image_button`

```
// Глава 03\Book02\sample3.dcl
sample3: dialog {label="Пример диалога с графическими элементами";
:image{key="b1";width=20; height=8;color=2;}
:image_button{key="b2"; action="(setq col 5)";color=5;
width=6;height=3;fixed_width=true;}
ok_button;
}// конец sample3
```

На рис. 3.5 показано формируемое диалоговое окно.

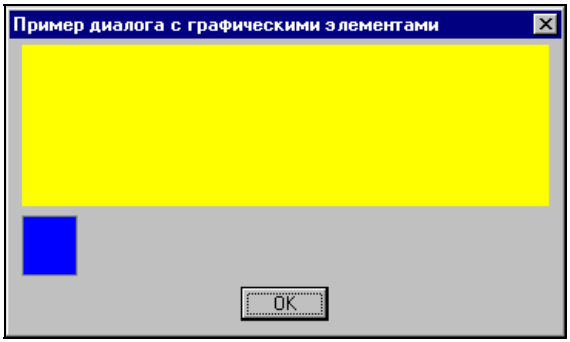


Рис. 3.5. Пример использования элементов `image` и `image_button`

Оба элемента имели бы одинаковую ширину, если бы во втором не был применен атрибут `fixed_width`. Цвет изображений задан атрибутом `color`.

С помощью функций `start_image`, `end_image` можно в вызывающей программе менять изображения или размещать на них слайды *(см. разд. 3.6)*.

3.3.4.4. Элементы со списками

Листинг 3.6 демонстрирует способ применения элемента `list_box`, который позволяет размещать в диалоговом окне данные в форме списка.

Листинг 3.6. Пример диалога с элементом `list_box`

```
// Глава 03\Book02\sample4.dcl
sample4: dialog {label="Пример диалога со списком";
:list_box{label="Список данных";key="b1";
list="Данное 1\nДанное 2\nДанное 3\nДанное 4";fixed_width_font=true;}
ok_button;
}// конец sample4
```

На рис. 3.6 показано диалоговое окно со списком. Значение `true` атрибута `multiple_select` предоставляет возможность пользователю с помощью клавиш <Shift> или <Ctrl> отметить в списке сразу несколько строк с данными. Значение `true` атрибута `fixed_width_font` заставляет систему AutoCAD выводить список, применяя тот шрифт, у которого ширина символов одинакова.

Если добавить к атрибутам элемента `list_box` задание атрибута `height` (`height=4;`), то список будет изображен только в три строки, что вызовет появление вертикальной полосы прокрутки (рис. 3.7).

С помощью функций `start_list`, `add_list` и `end_list` можно программно менять содержимое списка *(см. разд. 3.5)*.

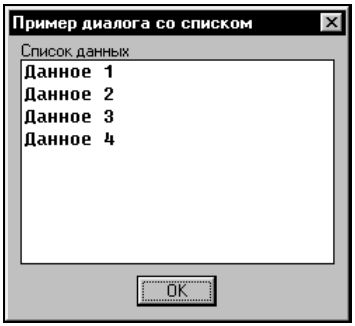


Рис. 3.6. Пример 1 использования элемента `list_box`

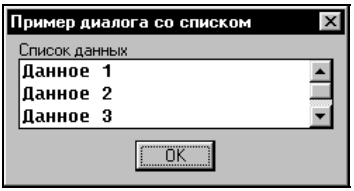


Рис. 3.7. Пример 2 использования элемента `list_box`

Листинг 3.7 демонстрирует способ применения элемента `popup_list`, который позволяет выводить в диалоговое окно данные в форме компактного раскрывающегося списка.

Листинг 3.7. Пример диалога с элементом `popup_list`

```
// Глава 03\Book02\sample5.dcl
sample5: dialog {label="Пример диалога с раскрывающимся списком";
:popup_list{label="Список данных";key="b1";
list="Данное 1\nДанное 2\nДанное 3\nДанное 4";fixed_width_font=true;}
ok_button;
}// конец sample5
```

На рис. 3.8 показано сформированное в листинге 3.7 диалоговое окно. Пользователю, для того чтобы раскрыть список, нужно щелкнуть мышью по треугольному значку в правой части списка.

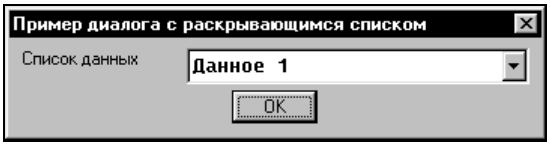


Рис. 3.8. Пример использования элемента `popup_list`

Для изменения из программы содержимого раскрывающегося списка применяются функции `start_list`, `add_list` и `end_list`.

3.3.4.5. Переключатели

Переключатели (радиокнопки), сгруппированные в горизонтальные радиоряды и вертикальные радиоколонки, дают возможность пользователю выбирать в диалоговом окне ровно одно значение из перечисленных около изображения радио-

кнопок. Переключателю соответствует элемент `radio_button`, а радиоряду и радиоколонке — элементы `radio_row` и `radio_column` соответственно. В листинге 3.8 приведен пример диалога со всеми этими элементами.

Листинг 3.8. Пример диалога с элементами `radio_button`, `radio_row` и `radio_column`

```
// Глава 03\Book02\sample6.dcl
sample6: dialog {label="Пример диалога с радиорядом и радиоколонкой";
:radio_column{label="Колонка выбора";key="b1";
  :radio_button{label="Использовать свойство 1"; key="b11";value="1";}
  :radio_button{label="Использовать свойство 2"; key="b12";value="0";}
  :radio_button{label="Использовать свойство 3"; key="b13";value="0";}
}
:radio_row{label="Ряд выбора";key="b2";
  :radio_button{label="Запад"; key="b21";value="0";}
  :radio_button{label="Север"; key="b22";value="0";}
  :radio_button{label="Восток"; key="b23";value="1";}
  :radio_button{label="Юг"; key="b24";value="0";}
}
ok_button;
}// конец sample6
```

Из листинга видно, что директивы описания элементов `radio_button` входят внутрь директив описания элементов `radio_row` и `radio_column`. На рис. 3.9 показано сформированное в листинге 3.8 диалоговое окно.

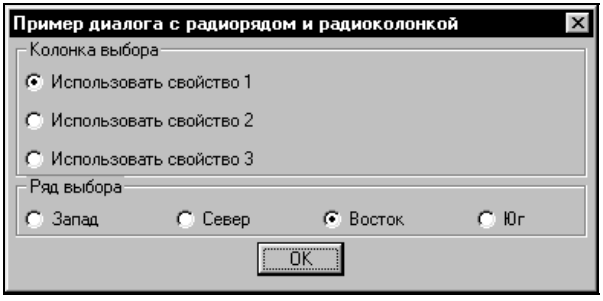


Рис. 3.9. Пример 1 использования элементов `radio_row` и `radio_column`

В элементе `radio_button` значение атрибута `value`, равное `"1"`, соответствует включенному переключателю, а `"0"` — выключенному. В радиоряду и радиоколонке только один переключатель может быть включен. Поэтому точки (символ выбора) на рис. 3.9 стоят только у первого переключателя в радиоколонке и у третьего — в радиоряду.

Другой способ задать включенный переключатель — указать имя его кода в качестве значения атрибута `value` элемента `radio_row` или `radio_column`. Например, если в листинге 3.8 элементу `radio_row` добавить `value="b12";`, то это означало бы то же самое, что переключателю с кодом `"b12"` задать `value="1";`.

На рис. 3.9 элементы `radio_row` и `radio_column` выглядят практически так же, как изображенные вместо них элементы `boxed_radio_row` и `radio_column` (радиоряд в рамке и радиоколонка в рамке).

Отличия возникают, когда у элементов не задан атрибут `label`, значение которого становится заголовком элемента. В этом случае у элементов `radio_row` и `radio_column` рамка не выводится (рис. 3.10).

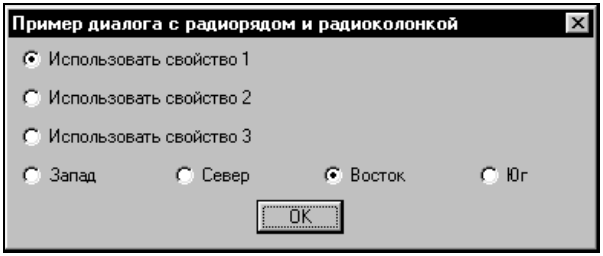


Рис. 3.10. Пример 2 использования элементов `radio_row` и `radio_column`

3.3.4.6. Скользящая шкала

Элемент со скользящей шкалой (`slider`) используется в тех случаях, когда диапазон выбираемых значений достаточно велик, но сами значения ограничены снизу и сверху. В листинге 3.9 приводится пример диалога с горизонтальным положением такого элемента.

Листинг 3.9. Пример диалога с элементом `slider`

```
// Глава 03\Book02\sample7.dcl
sample7: dialog {label="Пример диалога со скользящей шкалой";
:slider{key="b1";min_value=50;max_value=450;value=200;
layout=horizontal;small_increment=10;big_increment=50;}
ok_button;
}// конец sample7
```

На рис. 3.11 показано сформированное в листинге 3.9 диалоговое окно.

Ориентация шкалы (горизонтальная или вертикальная) управляется значением атрибута `layout`. Значения атрибутов `min_value` и `max_value` задают минимальное и максимальное значение из диапазона. Значения атрибутов `small_increment` и `big_increment` задают шаги изменения значений в различных ситуациях. В ка-

честве начального положения шкалы используется значение, заданное в атрибуте `value`.

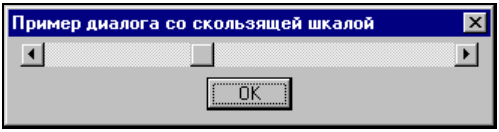


Рис. 3.11. Пример использования элемента `slider`

3.3.4.7. Флажок

Элементы управления с флажком описываются с помощью `toggle`. В листинге 3.10 приведен пример диалога с флажками.

Листинг 3.10. Пример диалога с элементом `toggle`

```
// Глава 03\Book02\sample8.dcl
sample8: dialog {label="Пример диалога с флажками";
:toggle{label="Сохранять контур";key="b1";value="1";}
:toggle{label="Разрешить изменение";key="b2";value="0";}
:toggle{label="Порождать вырезы";key="b3";value="1";}
ok_button;
}// конец sample8
```

На рис. 3.12 видно окно, сформированное в листинге 3.10. Значение атрибута `value` задает исходное положение переключателя: при `"1"` флажок установлен (у первого и третьего переключателей), при `"0"` — сброшен (у второго переключателя).

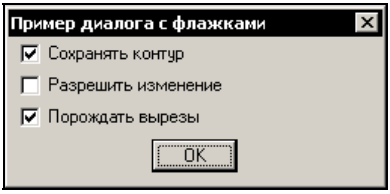


Рис. 3.12. Пример использования элемента `toggle`

3.3.5. Группировка полей

Из приведенных выше примеров видно, что по умолчанию система AutoCAD располагает объекты диалогового окна вертикально, один над другим. Такое расположение соответствует специальному оператору — `column` (колонка). Горизонтальному способу группировки полей соответствует оператор `row` (ряд).

Пример, приведенный в листинге 3.1, мог бы быть записан в следующем виде (см. листинг 3.11).

Листинг 3.11. Пример диалога с оператором `column`

```
// Глава 03\Book02\sample9.dcl
sample9: dialog{label="Настройка данных постели и таблицы";
:column{label="Настройка оборудования";children_alignment=centered;
  :edit_box{label="Ширина таблицы";key="kShir";value="750";edit_width=6;}
  :edit_box{label="Высота таблицы";key="kVys";value="300";edit_width=6;}
  :edit_box{label="Высота букв";key="kHbuk";value="180";edit_width=6;}
  :spacer{height=1;}
  :edit_box{label="Количество балок";key="kNumbalk";
            value="8";edit_width=6;}
  :edit_box{label="Количество коксов в балках";key="kNumkoks";
            value="17";edit_width=6;}
}// column
  :spacer{height=1;}
  :radio_row{label="Балки располагаются в таблице";
    :radio_button{label="по столбцам";key="kPoStolb";value="1";}
    :radio_button{label="по строкам";key="kPoStrok";value="0";}
  }
  :spacer{height=1;}
  ok_cancel;
}// конец sample9
```

Ключевое слово `column` означает, что элементы, директивы которых следуют внутри `column`, рассматриваются при выводе диалогового окна на экран как единое целое (в одну колонку). Значения таких атрибутов, как `children_alignment`, `children_fixed_width` и `children_fixed_height`, действуют по умолчанию на все подчиненные элементы колонки, у которых не заданы свои атрибуты `alignment`, `fixed_width` и `fixed_height`.

Оператор `column` может иметь свой заголовок (его текст задается значением атрибута `label`), как это приведено в листинге 3.11. Тогда колонка берется системой AutoCAD в рамку (в этом случае результат получается тот же, что и для оператора `boxed_column`).

Соответствующее листингу 3.11 диалоговое окно показано на рис. 3.13.

Если бы оператор `column` использовался без заголовка, то результат был бы таким же, как и на рис. 3.1.

Другой оператор, который группирует входящие в него элементы, — оператор `row`. Внутри него все подчиненные элементы располагаются по горизонтали. Соответствующий пример приведен в листинге 3.12.

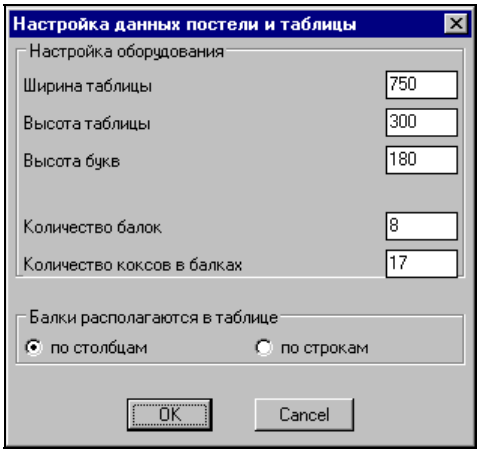


Рис. 3.13. Пример использования оператора `column`

Листинг 3.12. Пример диалога с оператором `row`

```
// Глава 03\Book02\sample10.dcl
sample10: dialog {label="Пример диалога с рядом кнопок";
:row{label="Ряд";
  :button{label="Кнопка 1"; key="b1"; action="(setq bstyle 1)";}
  :button{label="Кнопка 2"; fixed_width=true; alignment=right; key="b2";
          action="(setq bstyle 2)";}
}
ok_button;
}// конец sample10
```

На рис. 3.14 приведено формируемое диалоговое окно.

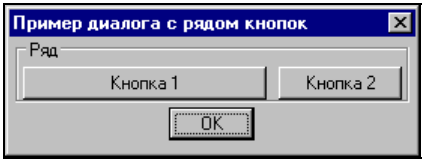


Рис. 3.14. Пример использования оператора `row`

Элементы, формируемые операторами `column` и `row`, не обводятся рамкой, если значение их атрибута `label` не задано или задано равным пустой строке.

Аналогичные функции выполняют операторы `boxed_column` и `boxed_row`, но формируемые ими элементы обводятся рамкой даже при отсутствии заголовка.

Уровней группировки может быть несколько, т. е. колонки и ряды сами могут быть элементами других колонок или рядов. Поэтому у разработчика приложе-

ний есть возможность формировать диалоговые окна сложной структуры. В листинге 3.13 и на рис. 3.15 приведен пример многоуровневой группировки.

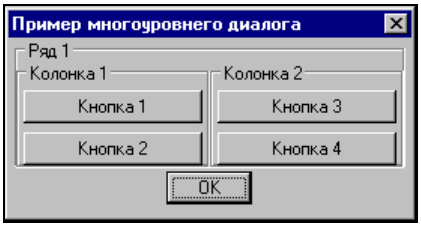


Рис. 3.15. Пример диалога с многоуровневой группировкой

Листинг 3.13. Пример многоуровневой группировки элементов

```
// Глава 03\Book02\sample11.dcl
sample11: dialog {label="Пример многоуровнего диалога";
:row{label="Ряд 1";
  :column{label="Колонка 1";
    :button{label="Кнопка 1";key="b1";}
    :button{label="Кнопка 2";key="b2";}
  }//column
  :column{label="Колонка 2";
    :button{label="Кнопка 3";key="b3";}
    :button{label="Кнопка 4";key="b4";}
  }//column
}//row
ok_button;
}// конец sample11
```

3.3.6. Элементы надписей

Самый употребительный элемент, который позволяет размещать в диалоговом окне надпись, — это элемент `text`. Он относится к оформительским. Текст, выведенный в элемент `text`, может быть изменен программно с помощью функции `set_tile`.

Иногда возникает необходимость формирования надписи из частей, которые располагаются в одном ряду друг с другом или в одной колонке. В этом случае можно пользоваться следующими операторами:

- `text_part` — часть текста, включаемая в элементы `concatenation` и `paragraph`;
- `concatenation` — горизонтальное соединение элементов `text_part`;
- `paragraph` — вертикальное соединение элементов `text_part` и `concatenation`.

Части надписей, создаваемые элементом `text_part`, могут изменяться программным путем с помощью функции `set_tile`.

В листинге 3.14 приведен пример сложной надписи, формируемой из элементов трех перечисленных типов.

Листинг 3.14. Пример диалога с элементами `text_part`, `concatenation` и `paragraph`

```
// Глава 03\Book02\sample12.dcl
sample12: dialog {label="Пример сложного текста";
:paragraph{key="p1";
  :concatenation{key="c1";
    :text_part{label="Три девицы";key="t1";}
    :text_part{label="под окном";key="t2";}
  }//concatenation
  :concatenation{key="c2";
    :text_part{label="Пряли";key="t3";}
    :text_part{label="поздно вечерком.";key="t4";}
  }//concatenation
}//paragraph
ok_button;
}// конец sample12
```

Результирующее диалоговое окно показано на рис. 3.16.

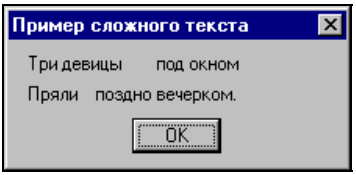


Рис. 3.16. Пример диалога с элементами `text_part`, `concatenation` и `paragraph`

3.3.7. Элементы промежутков

Для управления процессом выравнивания элементов в диалоговом окне можно пользоваться следующими операторами (элементами промежутков), которые не содержат отображаемых элементов оформления или элементов управления:

- `spacer` — элемент произвольной ширины и высоты (нужно обязательно задать атрибуты `width` и `height`);
- `spacer_0` — элемент `spacer` с нулевыми значениями атрибутов `width` и `height`;
- `spacer_1` — элемент `spacer` с единичными значениями атрибутов `width` и `height`.

Два последних элемента являются частными случаями первого, но для удобства выделены в отдельные элементы.

Все эти элементы резервируют для себя место в рядах или колонках, задавая тем самым дополнительный сдвиг остальных элементов. Вернемся к примеру из листинга 3.13 и изменим его, добавив элемент `spacer` с двойным промежутком в вертикальную колонку между основными кнопками и кнопкой выхода и элемент `spacer` с единичным промежутком — в горизонтальный ряд между колонками (см. листинг 3.15 и рис. 3.17).

Листинг 3.15. Пример диалога с элементом `spacer`

```
// Глава 03\Book02\sample13.dcl
sample13: dialog {label="Пример диалога с промежутком";
:row{label="Ряд 1";
:column{label="Колонка 1";
 :button{label="Кнопка 1";key="b1";}
 :button{label="Кнопка 2";key="b2";}
}//column
spacer_1;
:column{label="Колонка 2";
 :button{label="Кнопка 3";key="b3";}
 :button{label="Кнопка 4";key="b4";}
}//column
}//row
:spacer{width=0;height=2;}
ok_button;
}// конец sample13
```

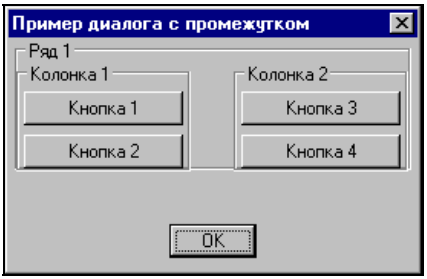


Рис. 3.17. Пример диалога с элементом `spacer`

По сравнению с рис. 3.15 видно улучшение читаемости диалогового окна за счет задания дополнительных промежутков.

3.4. Функции, работающие с элементами диалога

Несколько функций языка LISP управляют элементами и их действиями. Они позволяют:

- задавать LISP-выражение, выполняемое при выборе элемента и нажатии клавиши <Enter> или щелчке левой кнопкой мыши (функция `action_tile`);
- отключать (гасить) элемент и включать его обратно (функция `mode_tile`);
- читать текущее содержимое элемента (функция `get_tile`);
- менять текущее содержимое элемента (функция `set_tile`);
- читать значение атрибута элемента (функция `get_attr`);
- связывать с элементом данные пользователя (функция `client_data_tile`).

3.4.1. Функция *action_tile*

Функция `action_tile` задает при помощи LISP-выражения действие, которое нужно выполнить системе AutoCAD при выборе элемента или редактировании его содержимого. Синтаксис функции:

```
(action_tile <код> <выражение>)
```

Аргументы: `<код>` — строка с кодом элемента, `<выражение>` — строка с LISP-выражением.

Возвращаемое значение — `T`.

Для большей части элементов диалогового окна, являющихся элементами управления, должны быть описаны действия, выполняемые системой AutoCAD при выборе пользователем этого элемента. Под выбором элемента следует иметь в виду его выделение и нажатие клавиши <Enter> или щелчок мышью для элемента `button`, редактирование содержимого поля `edit_box`, выбор элемента (элементов) списка `list_box` или `popup_list`, движение шкалы для `slider` и т. п. Действия выполняются, когда после операции с элементом (редактирования, выбора элемента списка и т. п.) курсор перейдет к другому элементу или пользователь выйдет из диалогового окна с помощью кнопки выхода.

Такое действие для каждого элемента может быть задано и с помощью атрибута `action` DCL-файла. Значением атрибута является строка с LISP-выражением (выражение может быть длинным, но не должно использовать функцию `command`). Другой вариант задания действия — указание LISP-выражения в строке, передаваемой в качестве аргумента `<выражение>` функции `action_tile`, обращение к которой в программе должно быть записано между вызовами функций `new_dialog` и `start_dialog`.

Для тех элементов диалогового окна, для которых действие не указано ни в атрибуте action, ни в функции action_tile, используется аргумент <действие>, заданный в вызове функции new_dialog.

Действие, определенное в функции action_tile с помощью аргумента <выражение>, замещает (аннулирует) действие, заданное выражением (значением) атрибута action этого элемента в DCL-файле.

Действие всегда задается через код элемента, к которому оно относится. Код элемента — это значение атрибута key этого элемента. Именно этот код и передается функции action_tile в качестве аргумента <код>.

LISP-выражение в функции action_tile может использовать значения зарезервированных глобальных переменных со следующими именами:

- $value — текущее значение рассматриваемого элемента;
- $key — код рассматриваемого элемента;
- $reason — тип действия, выполненного над элементом;
- $data — данные элемента, заданные функцией client_data_tile в приложении пользователя;
- $x и $y — координаты указанной пользователем точки внутри изображения (только для элементов с изображениями).

О зарезервированных переменных *см. разд. 3.4.7.*

Примеры:

- (action_tile "accept" "(done_dialog)") — пример действия кнопки выхода;
- (action_tile "Drt1" (strcat "(if (<= (atoi $value) 0)" "(alert \"Отрицательные числа не допускаются!\")")) — пример проверки значения элемента на отрицательные числа или нуль.

В последнем примере длинное LISP-выражение, задаваемое в качестве действия для элемента с кодом "Drt1", получено с помощью сцепки двух строк. Обратите внимание на то, что внутренние кавычки, необходимые в выражении для вызова функции alert, применяются с предшествующим символом \.

3.4.2. Функция *mode_tile*

Функция mode_tile изменяет состояние элемента диалога. Синтаксис:

```
(mode_tile <код> <состояние>)
```

Аргументы: <код> — код элемента; <состояние> — целое число, от 0 до 4 (см. табл. 3.4).

Возвращаемое значение — nil.

При выключении элемента его цвет приглушается и он становится недоступным для пользователя. Для того чтобы этот элемент стал доступным, его нужно в дальнейшем включить.

Таблица 3.4. *Значения аргумента* `<состояние>` *функции* `mode_tile`

Значение	Действие
0	Включить элемент
1	Выключить (сделать недоступным) элемент
2	Выбрать элемент
3	Подсветить содержимое элемента
4	Включить/выключить подсветку изображения

Примеры:

- `(mode_tile "k1" 1)` выключает элемент с кодом `"k1"` (оно становится недоступным);
- `(mode_tile "k1" 0)` включает элемент с кодом `"k1"` (оно снова становится доступным).

3.4.3. Функция *set_tile*

Функция `set_tile` позволяет менять содержимое элемента диалога, заданное атрибутом `value`. Синтаксис:

```
(set_tile <код> <значение>)
```

Аргументы: `<код>` — код элемента; `<значение>` — строка с новым значением, присваиваемым атрибуту `value` указанного элемента.

Возвращаемое значение — значение второго аргумента, переданного функции.

3.4.4. Функция *get_tile*

Функция `get_tile` читает содержимое элемента диалога (текущее значение атрибута `value`). Синтаксис:

```
(get_tile <код>)
```

Аргумент `<код>` — строка с кодом элемента.

Возвращаемое значение — строка с текущим значением атрибута `value`.

Пример:

- `(get_tile "usfam")` возвращает значение элемента с кодом `"usfam"`.

3.4.5. Функция *get_attr*

Функция `get_attr` читает значение атрибута элемента текущего диалога. Синтаксис:

```
(get_attr <код> <атрибут>)
```

Аргументы: <код> — строка с кодом элемента, <атрибут> — наименование атрибута.

Возвращаемое значение — строка со значением атрибута выбранного элемента.

Пример:

- (get_attr "usfam" "alignment") для элемента с кодом "usfam" возвращает значение его атрибута "alignment" (например, "centered").

3.4.6. Функция *client_data_tile*

Функция client_data_tile связывает с элементом диалогового окна строку с пользовательскими данными. Синтаксис:

```
(client_data_tile <код> <данные>)
```

Аргументы: <код> — строка с кодом элемента; <данные> — строка с пользовательскими данными.

Возвращаемое значение — nil.

Обратиться к данным, присоединенным к элементу диалогового окна, можно, прочитав значение переменной $data в функции action_tile. Таким образом можно изменять действие, присвоенное элементу, в зависимости от пользовательских данных.

Пример:

- (client_data_tile "usfam" "2110") связывает строку "2110" с элементом "usfam".

3.4.7. Переменные действия

Во время работы диалогового окна система AutoCAD формирует значения глобальных переменных $key, $value, $reason, $data, $x, $y (они называются *переменными действия*), сохраняя в них некоторые дополнительные данные о состоянии этого окна и выполненных в нем действиях пользователя. Переменные используются в LISP-выражениях действия, задаваемых в функциях action_tile, new_dialog и атрибуте action (вне выражений действия значения переменных не определены).

Пользовательская программа не может изменять значения этих переменных и должна применять их только для чтения. В табл. 3.5 рассматриваются различные варианты формирования значений переменных действия.

Примеры:

- (action_tile "usfam" "(setq nam (strcat \"Пользователь: \" $value))");
- (action_tile "cldat" "(setq tyg (read $data))").

Таблица 3.5. *Переменные действия и варианты их значений*

Переменная	Значение
`$key`	Код элемента (во всех элементах)
`$value`	Значение атрибута `value` (во всех элементах). Для поля `edit_box` — это введенная строка; для `toggle` — `"1"` (включен) или `"0"` (выключен); для `list_box` и `popup_list` — строка с номером (или номерами) выбранных элементов списка (`nil`, если ни один элемент списка не выбран)
`$data`	Строка с пользовательскими данными, привязанными к элементу *(см. предыдущий раздел)*
`$reason`	Для элементов `edit_box`, `list_box`, `image_button` и `slider` — число с кодом причины прерывания работы с элементом (подробнее см. табл. 3.6)
`$x, $y`	Координаты (в пикселах) указанной пользователем точки внутри изображения для элемента `image_button`

В табл. 3.6 приведена дополнительная информация о значениях переменной `$reason`.

Таблица 3.6. *Значения переменной* `$reason`

Значение	Описание
1	Обычный вариант действия с элементом, в том числе: пользователь выбрал элемент; нажата клавиша <Enter>, а элемент имеет значение `true` атрибута `is_default`
2	Для поля `edit_box`: пользователь вышел из поля, но не завершил редактирование (нажал клавишу <Tab> или выбрал другой элемент)
3	Для элемента `slider`: пользователь изменил значение элемента с помощью указателя шкалы, но не завершил ввода
4	Для элементов `list_box` и `image_button`: пользователь выполнил двойной щелчок (если `multiple_selection=false`, то это должно означать окончательный выбор, если `multiple_selection=true`, то это должно обрабатываться как обычный выбор — см. значение 1)

3.4.8. Пример

Для диалогов и текстовых полей значения атрибута `value` заменяют заголовки (значения атрибута `label`). Рассмотрим диалог, приведенный в листинге 3.16.

Внешний вид диалогового окна, соответствующего диалогу в листинге 3.16, приведен на рис. 3.18.

Листинг 3.16. Пример диалога `user` (ввод фамилии пользователя)

```
// Глава 03\Book03\user.dcl
user:dialog{label = "Ввод фамилии пользователя";
    :column{
      :edit_box{label="Фамилия:";edit_width=40; alignment=centered;
        value=""; key="usfam";
      }// конец edit_box
      :spacer{height=1;}
      :text{label="Выбор из списков подразделений";alignment = centered;}
      :row{
        :text{label="Подразделение 0001"; key="s1";}
        :popup_list{key="p1";is_enabled=true;label="";list="";width=20;}
      }// конец row
      :row{
        :text{label="Подразделение 0002"; key="s2";}
        :popup_list{key="p2";is_enabled=true;label="";list="";width=20;}
      }// конец row
      :row{
        :text{label="Подразделение 0003"; key="s3";}
        :popup_list{key="p3";is_enabled=true;label="";list="";width=20;}
      }// конец row
      :row{
        :text{label="Подразделение 0004"; key="s4";}
        :popup_list{key="p4";is_enabled=true;label="";list="";width=20;}
      }// конец row
   :spacer {height=1;}
    ok_cancel;
  }// конец column
}// конец dialog
```

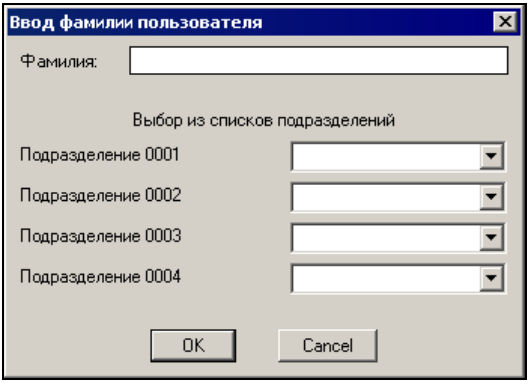


Рис. 3.18. Диалоговое окно **Ввод фамилии пользователя**

Это диалоговое окно предназначено для ввода фамилии пользователя прикладной системы в поле **Фамилия** (`edit_box`) или выбора фамилии из списков сотрудников подразделений, причем наименования подразделений могут меняться. Максимальное количество подразделений — 4 (диалог легко может быть расширен на любое количество). Если количество подразделений в данный момент меньше четырех, то лишние строки в окне нужно выключить.

В начальном состоянии диалогового окна, приведенного на рис. 3.18, списки сотрудников пусты, а в качестве наименований подразделений даны условные наименования **Подразделение 0001**, **...**, **Подразделение 0004**. Наименования подразделений должны быть достаточно длинными, поскольку при определении размеров выводимых элементов система AutoCAD ориентируется на значения атрибутов, заданных в DCL-файле. Замена наименований на фактические будет произведена программно с помощью функции `set_tile`. Если заменяющие наименования будут длиннее замененных, то они будут урезаться системой AutoCAD.

Кроме того, программа (см. далее) должна читать текстовые файлы со списками подразделений, расположенные в специальной папке, и по количеству файлов затем определять количество подразделений. Текстовый файл со списком сотрудников будет выглядеть примерно так, как это приведено в листинге 3.17.

Листинг 3.17. Файл polz01.txt

```
Сектор 2111
Алексеева
Борисов
Витольд
Савельев
Тверской
Юдинцева
Яковлева
```

В этом файле первая строка задает наименование подразделения, а следующие строки — фамилии сотрудников в алфавитном порядке.

В листинге 3.18 приведен текст программы (функции `usn`), работающей с диалогом `user` (см. листинг 3.16).

Листинг 3.18. Функция `usn`

```
; Глава 03\Book03\usn.lsp
(defun usn ( / dcl_id fpolz fd i plist nlist spi str nsubd)
(if (not pv)(setq pv ""))
(if (< (setq dcl_id (load_dialog "user.dcl")) 0)(exit))
(if (not (new_dialog "user" dcl_id)) (exit))
(setq i 0)
```

```
; Чтение файлов polz01.txt, polz02.txt, polz03.txt, polz04.txt
(repeat 4; количество подразделений — 4
  (setq spi nil)
  (setq i (1+ i) fpolz (strcat "polz0" (itoa i) ".txt"))
  (if (findfile fpolz)
    (progn
      (setq fd (open (findfile fpolz) "r"))
      (setq str (read-line fd))
      (setq nlist (append nlist (list str)))
      (if str
        (while (setq str (read-line fd))
          (setq spi (append spi (list str)))
        );while
      );if str
      (close fd)(setq fd nil)
    );progn
    (setq nlist (append nlist (list nil)))
  );if findfile
  (setq plist (append plist (list spi)))
);repeat
;
; Заполнение списков диалогового окна
(setq i -1)
(repeat 4
  (setq i (1+ i))
  (if (setq nsubd (nth i nlist))
    (progn
      (set_tile (strcat "s" (itoa (1+ i))) nsubd)
      (if (setq spi (nth i plist))
        (progn
          (start_list (strcat "p" (itoa (1+ i))))
          (mapcar 'add_list spi)
          (end_list)
        );progn
      );if
    );progn
; Гашение списка, если файл с фамилиями не найден
    (progn
      (mode_tile (strcat "s" (itoa (1+ i))) 1)
      (mode_tile (strcat "p" (itoa (1+ i))) 1)
    )
  );if
);repeat
; Элементы управления диалогового окна
(action_tile "accept"
```

```
  (strcat
    "(if(= (setq pv(get_tile \"usfam\"))\"\")"
    "(alert\"Фамилия не задана\")"
    "(done_dialog))"
  );strcat
);action-tile accept
(action_tile "cancel" "(done_dialog)")
(action_tile "p1"
  "(set_tile \"usfam\" (nth (atoi $value) (nth 0 plist)))")
(action_tile "p2"
  "(set_tile \"usfam\" (nth (atoi $value) (nth 1 plist)))")
(action_tile "p3"
  "(set_tile \"usfam\" (nth (atoi $value) (nth 2 plist)))")
(action_tile "p4"
  "(set_tile \"usfam\" (nth (atoi $value) (nth 3 plist)))")
(start_dialog)
(unload_dialog dcl_id)
(princ)
);defun usn
```

Эта программа с помощью функции `set_tile`, входящей внутрь LISP-выражений, записанных как действие для элементов с кодами `"p1"`, `"p2"`, `"p3"` и `"p4"`, заменяет содержимое списков этих элементов на списки, прочитанные из файлов polz01.txt, polz02.txt, polz03.txt, polz04.txt с фамилиями сотрудников подразделений (использованные в этой операции функции `start_list`, `add_list` и `end_list` рассмотрены в *разд. 3.5*). Фамилии сотрудников накапливаются в списке `plist`, а наименования подразделений — в списке `nlist`.

Переменная `$value`, используемая функцией `action_tile`, для раскрывающегося списка `popup_list` принимает значение с номером выбранного пользователем элемента списка.

С помощью той же функции `set_tile` стандартные наименования подразделений (**Подразделение 0001** и т. д.) меняются на элементы списка `nlist`. Если количество подразделений оказалось меньше четырех (например, два), то элементы, связанные с лишними подразделениями, отключаются с помощью функции `mode_tile`.

Фамилия, либо введенная в поле `edit_box`, либо скопированная в это поле после выбора ее в одном из четырех раскрывающихся списков, заносится программой в глобальную переменную `pv`, которую дальше может использовать приложение.

На рис. 3.19 приведено диалоговое окно **Ввод фамилии пользователя**, открываемое функцией `usn`, после того как пользователь выбрал в первом подразделении первую фамилию. Если пользователь нажмет кнопку **OK** в тот момент, когда фамилия еще не введена или не выбрана с помощью раскрывающегося списка

(т. е. поле с кодом "usfam" останется пустым), то программа выведет сообщение "Фамилия не задана".

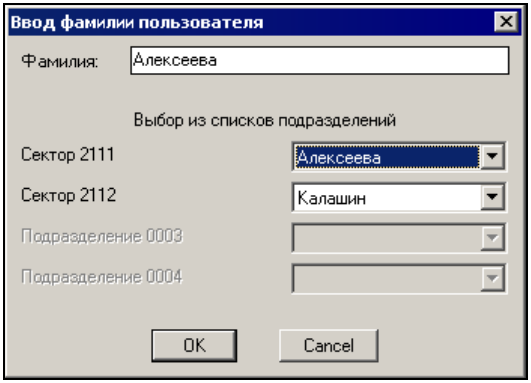


Рис. 3.19. Диалоговое окно после выбора фамилии

3.5. Функции, работающие со списками

Для формирования списков, элементы которых отображаются в list_box и popup_list, применяются специальные функции start_list, add_list и end_list.

3.5.1. Функция *start_list*

Функция start_list начинает операцию формирования или редактирования списка в list_box или popup_list. Синтаксис функции:

```
(start_list <код> [<операция> [<номер>]])
```

Аргументы: <код> — код элемента, список которого редактируется; <операция> — целое число, обозначающее тип операции (1 — изменить выбранные элементы; 2 — добавить новые элементы; 3 — удалить старый список и создать новый); <номер> — номер элемента, начиная с которого вносятся изменения (используется, если значение аргумента <операция> равно 1). По умолчанию для второго аргумента действует значение 3, для третьего — 0. Элементы в списке нумеруются, начиная с нуля.

Возвращаемое значение — код элемента, список которого будет редактироваться.

Вызов функции start_list является отметкой начала операций формирования или редактирования списка, вызов функции end_list — отметкой окончания. Изменения вносятся с помощью функции add_list и других функций, располагающихся между start_list и end_list. Не допускается между start_list и end_list использовать функцию set_tile.

Примеры:

- `(start_list "kp2")` указывает начало операции формирования заново списка для элемента диалога с кодом `"kp2"`;
- `(start_list "kp2" 2)` указывает начало операции добавления новых элементов списка для элемента диалога с кодом `"kp2"`;
- `(start_list "kp2" 1 14)` указывает начало операции замены элементов списка, начиная с 14, для элемента диалога с кодом `"kp2"`.

3.5.2. Функция *add_list*

Функция `add_list` выполняет операцию редактирования списка в `list_box` или `popup_list`. Синтаксис:

```
(add_list <строка>)
```

Аргумент `<строка>` — строка, которая добавляется к списку или заменяет элемент списка.

Возвращаемое значение — значение аргумента `<строка>`, если операция выполнена успешно. При сбое возвращается `nil`.

Тип операции (изменение или добавление), выполняемой функцией `add_list`, определяется предшествующим вызовом функции `start_list`.

Примеры:

- `(add_list "Деталь 3217-06")`
- `(mapcar 'add_list detss)`

В последнем случае применена функция `mapcar`, что особенно удобно, когда количество элементов списка заранее не известно. Список `detss` в конкретном случае может иметь, например, вид `(list "Деталь 12-05" "Деталь A14" "Деталь 7-29")`.

3.5.3. Функция *end_list*

Функция `end_list` завершает операцию редактирования списка в `list_box` или `popup_list`. Синтаксис:

```
(end_list)
```

Возвращаемое значение — `nil`.

3.5.4. Пример окна со списками

Пример окна, использующего четыре элемента `popup_list`, был приведен выше (см. листинг 3.16 и рис. 3.18). Начальное состояние каждого из этих списков, имеющих коды `"p1"`, `"p2"`, `"p3"` и `"p4"`, задано в DCL-файле user.dcl: `list="";`.

Формирование списков выполняется с помощью функций `start_list`, `add_list` и `end_list` после прочтения соответствующих файлов с фамилиями сотрудников подразделений. При заполнении списков функция `add_list` использована в сочетании с функцией `mapcar`.

3.5.5. Ограничения списков в DCL-файлах

Начальные значения списка и раскрывающегося списка при заполнении их прямо в DCL-файле имеют ограничения по общей длине в 255 символов. Рассмотрим следующий пример (листинг 3.19).

Листинг 3.19. Пример диалога с длинным списком

```
// Глава 03\Book04\long_popuplist.dcl
long: dialog {label="Пример диалога с длинным списком";
:popup_list{label="Список данных";key="plist";
list=
"1_1234567891011\n2_1234567891011\n3_1234567891011\n4_1234567891011\n
5_1234567891011\n6_1234567891011\n7_1234567891011\n8_1234567891011\n
9_1234567891011\n10_123456789101\n11_123456789101\n12_123456789101\n
13_123456789101\n14_123456789101\n15_123456789101\n16_123456789101\n
17_123456789101\n18_123456789101\n19_123456789101";}
ok_button;
}//long
```

Центральным элементом диалогового окна является раскрывающийся список с кодом `"plist"`. В листинге 3.19 начальное заполнение списка, который должен иметь 19 элементов по 15 символов, осуществлено прямо в DCL-файле. Для этого атрибуту `list` в качестве значения задана строка `"1_1234567891011\n ... 19_123456789101"` (символ `"\n"` является разделителем будущих элементов списка). Длина такой строки получается равной $15 \times 19 = 285$ знакам. На рис. 3.20 показано, какое же состояние списка мы получим в таком варианте.

Мы видим, что на самом деле получили только 16 элементов (строк) длиной по 15 символов, т. е. всего 240 вместо 285.

Такой результат вызван ограничением, которое существует на длину строковых значений, применяемых в написании DCL-файлов (не более 255, включая символы перевода строки!).

Каков же выход? Не надо было заполнять раскрывающийся список прямо в DCL-файле (можно было написать `list="";`). Формирование списка лучше было перенести в LISP-программу, в которой можно использовать разные варианты преодоления ограничений на длину строк и т. п. В листинге 3.20 приведен пример заполнения длинного списка.

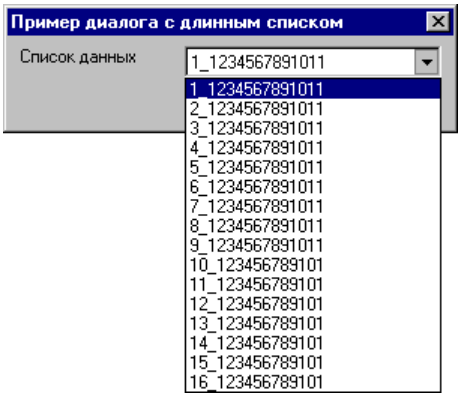


Рис. 3.20. Диалоговое окно с усеченным длинным списком

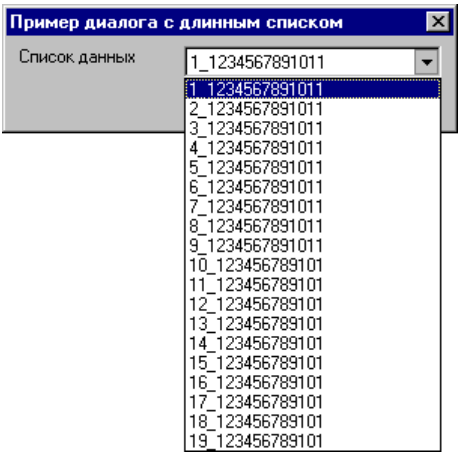


Рис. 3.21. Диалоговое окно с неусеченным длинным списком

Листинг 3.20. Заполнение диалогового окна с длинным списком

```
; Глава 03\Book04\long_popuplist.lsp
(defun test ( / )
(setq dcl_id (load_dialog "long_popuplist.dcl"))
(new_dialog "long" dcl_id)
(start_list "plist")
(mapcar 'add_list (list "1_1234567891011" "2_1234567891011"
  "3_1234567891011" "4_1234567891011" "5_1234567891011"
  "6_1234567891011" "7_1234567891011" "8_1234567891011"
  "9_1234567891011" "10_123456789101" "11_123456789101"
  "12_123456789101" "13_123456789101" "14_123456789101"
  "15_123456789101" "16_123456789101" "17_123456789101"
  "18_123456789101" "19_123456789101"))
(end_list)
(start_dialog)
(unload_dialog dcl_id)
); defun test
```

Если выполнить функцию `test`, которая открывает диалоговое окно, описанное в листинге 3.19, то раскрывающийся список уже не будет обрезан. На рис. 3.21 видно, что в нем как раз 19 элементов по 15 символов каждый.

3.5.6. Особенности применения функций *set_tile* и *get_tile* для списков

Следует обратить внимание еще на одну особенность списков и раскрывающихся списков. Она связана с тем, какое значение возвращает функция `get_tile` для

списка, в котором выбрана какая-то строка. Ошибкой было бы считать, что таким значением является текст выбранного элемента. Возвращается строка с номером выбранного элемента или номерами, если список допускает множественный выбор (для раскрывающегося списка это невозможно), например: "0", "4", "0 5", "1 2 3 4" и т. д. Поэтому необходимо где-то в памяти хранить еще LISP-список всех значений, чтобы из него по номеру извлечь нужный компонент.

Такое же замечание относится к функции set_tile, а также к переменной действия $value и атрибуту value.

Вернемся еще раз к примеру из листинга 3.18. В нем заполнение поля с кодом "usfam" после того, как пользователь в каком-то из списков сделал выбор, осуществляется с помощью следующего выражения (на примере обработки раскрывающегося списка с кодом "p1"):

```
(action_tile "p1"
  "(set_tile \"usfam\" (nth (atoi $value) (nth 0 plist)))")
```

Переменная plist здесь является хранителем полного перечня значений в виде LISP-списка. Поэтому второй вызов функции nth ((nth 0 plist)) вычисляет подсписок с номером 0, а первый вызов той же функции извлекает из подсписка нужное значение, т. к. переменная $value содержит в строковом виде номер выбранного в раскрывающемся списке элемента.

3.6. Функции, работающие с графическими элементами

Элементы image и image_button являются графическими элементами с прямоугольной зоной, которая может быть заполнена одним цветом, слайдом или отрезками разных цветов. Этой цели служат рассматриваемые далее функции start_image (открывает процесс заполнения), end_image (завершает процесс заполнения), а также функции fill_image, slide_image, vector_image.

3.6.1. Функция *start_image*

Функция start_image открывает процесс заполнения графического элемента. Синтаксис:

```
(start_image <код>)
```

Аргумент <код> — код графического элемента (значение атрибута key).

Возвращаемое значение — строка со значением аргумента. При сбое возвращается nil.

Вызов функции start_image является отметкой начала операций формирования изображения в графическом элементе, вызов функции end_image — отметкой

окончания. Формирование изображения выполняется с помощью функций `slide_image`, `fill_image`, `vector_image` и других функций, располагающихся между `start_image` и `end_image`. Не допускается между `start_image` и `end_image` использовать функцию `set_tile`.

Пример:

- `(start_image "gra4")`

3.6.2. Функция *end_image*

Функция `end_image` завершает процесс формирования изображения графического элемента. Синтаксис:

```
(end_image)
```

Возвращаемое значение — `nil`.

Вызову функции `end_image` должен обязательно предшествовать вызов функции `start_image`.

3.6.3. Функция *fill_image*

Функция `fill_image` заполняет прямоугольник внутри графического элемента одним цветом. Синтаксис:

```
(fill_image <x1> <y1> <ширина> <высота> <цвет>)
```

Аргументы: `<x1>`, `<y1>` — координаты (в пикселах) левого верхнего угла заполняемого прямоугольника; `<ширина>`, `<высота>` — ширина и высота (в пикселах) заполняемого прямоугольника; `<цвет>` — целое число с номером цвета, выбранного для заполнения. В качестве номера цвета можно использовать стандартные номера цветов системы AutoCAD от 0 до 256, а также некоторые отрицательные значения, приведенные в табл. 3.7.

Возвращаемое значение — целое число (значение аргумента `<цвет>`).

Координаты графического элемента определяются следующим образом. Левый верхний угол имеет координаты (0, 0), а координаты правого нижнего угла могут быть получены с помощью функций `dimx_tile` и `dimy_tile` (см. далее). Функция `fill_image` может заполнить одним цветом как весь графический элемент, так и любой прямоугольник внутри него.

В табл. 3.7 приведены номера специальных цветов, которые можно использовать в качестве отрицательных значений пятого аргумента функции.

Пример:

- `(fill_image 5 6 15 25 1)` заполняет красным цветом (номер 1) прямоугольник с левым верхним углом в точке (5, 6) и нижним правым углом (15, 25).

Таблица 3.7. *Номера специальных цветов, используемые функцией* `fill_image`

Номер	Описание
–2	Цвет фона графического экрана системы AutoCAD
–15	Цвет фона диалогового окна
–16	Цвет текста диалогового окна
–18	Цвет контура диалогового окна

3.6.4. Функция *slide_image*

Функция `slide_image` заполняет графический элемент слайдом из отдельного SLD-файла или из библиотеки слайдов. Синтаксис:

```
(slide_image <x1> <y1> <ширина> <высота> <слайд>)
```

Аргументы `<x1>`, `<y1>`, `<ширина>` и `<высота>` имеют тот же смысл, что и для функции `fill_image`. Аргумент `<слайд>` — имя файла слайда (расширение sld не указывается) или имя библиотеки слайдов (имя слайда из библиотеки указывается в круглых скобках).

Возвращаемое значение — строка (значение аргумента `<слайд>`).

Примеры:

- `(slide_image 0 0 50 32 "ring1")` заполняет графический элемент размером 50×32 слайдом из файла ring1.sld;
- `(slide_image 0 0 50 32 "lib23(nut8)")` заполняет графический элемент размером 50×32 слайдом nut8 из библиотеки слайдов lib23.

3.6.5. Функция *vector_image*

Функция `vector_image` рисует отрезок заданного цвета внутри графического элемента диалогового окна. Синтаксис:

```
(vector_image <x1> <y1> <x2> <y2> <цвет>)
```

Аргументы: `<x1>`, `<y1>` — координаты (в пикселах) начала рисуемого отрезка; `<x2>`, `<y2>` — координаты (в пикселах) конца отрезка; `<цвет>` — целое число с номером цвета от 0 до 256, или с отрицательным номером из табл. 3.7.

Возвращаемое значение — целое число (значение аргумента `<цвет>`).

Координаты, используемые функцией `vector_image`, вычисляются так же, как в описании функции `fill_image`.

Пример:

- `(vector_image 5 11 33 11 3)` рисует отрезок зеленого цвета (номер 3) из точки (5, 11) в точку (33, 11).

3.6.6. Функция *dimx_tile*

Функция `dimx_tile` вычисляет горизонтальный размер графического элемента с данным кодом (точнее, размер элемента минус 1). Синтаксис функции:

```
(dimx_tile <код>)
```

Аргумент `<код>` — строка с кодом графического элемента.

Возвращаемое значение — целое число с номером крайнего правого пиксела элемента (левый крайний пиксел имеет номер 0).

Пример:

- `(dimx_tile "igp2")` может вернуть 50 (всего в таком элементе по горизонтали будет 51 пиксел).

3.6.7. Функция *dimy_tile*

Функция `dimy_tile` вычисляет вертикальный размер графического элемента с данным кодом (точнее, размер элемента минус 1). Синтаксис:

```
(dimy_tile <код>)
```

Аргумент `<код>` — строка с кодом графического элемента.

Возвращаемое значение — целое число с номером крайнего нижнего пиксела элемента (верхний крайний пиксел имеет фиксированный номер 0).

Пример:

- `(dimy_tile "igp2")` может вернуть 32 (всего в таком элементе по вертикали будет 33 пиксела).

3.6.8. Примеры окон с графическими элементами

Некоторые примеры уже были приведены в *разд. 3.3.4.3*. Здесь будут показаны варианты более сложной обработки. Еще два примера можно найти в *приложениях 3* и *4* (индикатор и светофор).

3.6.8.1. Просмотр изображений из библиотеки слайдов

В листинге 3.21 размещено описание диалога с изображением слайда, которое можно изменять, выбирая в списке имя другого слайда.

Листинг 3.21. Просмотр слайдов

```
// Глава 03\Book05\slide.dcl
slide: dialog{label="Просмотр слайдов из библиотеки";
```

```
:row{
  :column{label="Слайд";
    :image{key="sld1";color=-2;width=32;height=9.5;}
  }//column
  :column{label="Список изображений";
    :list_box{key="names";list="";width=24;height=6.5;}
  }//column
}//row
spacer_1;
ok_button;
}//slide
```

Диалоговое окно, описанное в листинге 3.21, показано на рис. 3.22. Отображен первый слайд из библиотеки.

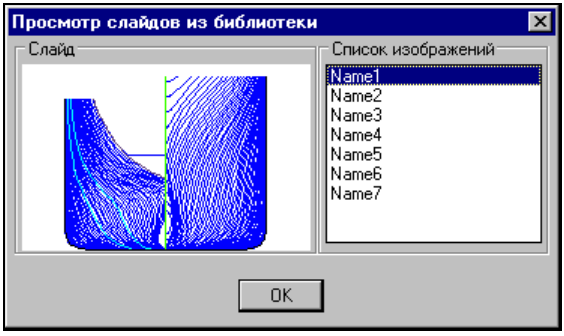


Рис. 3.22. Диалоговое окно с первым слайдом

Функции, управляющие работой этого окна, приведены в листинге 3.22.

Листинг 3.22. Управление сменой слайда

```
; Глава 03\Book05\slide.lsp
; Функция загрузки окна
(defun slide ( / dcl_id all dx dy)
(setq dcl_id (load_dialog "slide.dcl"))
(new_dialog "slide" dcl_id)
; Имена изображений
(setq all
  (list "Name1" "Name2" "Name3" "Name4" "Name5" "Name6" "Name7"))
(start_list "names")
(mapcar 'add_list all)
(end_list)
; Отметка первого в списке имени
(set_tile "names" "0")
```

```
; Размеры левого графического элемента
(setq dx (dimx_tile "sld1") dy (dimy_tile "sld1"))
; Загрузка первого слайда
(start_image "sld1")
(slide_image 0 0 dx dy "My(Name1)")
(end_image)
; Обработка выбора имени в списке
(action_tile "names" "(change_s $value all dx dy)")
(start_dialog)
(unload_dialog dcl_id)
); defun slide
; Функция смены слайда
(defun change_s (sn nlist x y / i name)
  (setq i (atoi sn) name (nth i nlist))
  ; Замена слайда
  (start_image "sld1")
  (fill_image 0 0 x y -2)
  (slide_image 0 0 x y (strcat "My(" name ")"))
  (end_image)
); defun change_s
```

Функция `slide` управляет загрузкой диалогового окна, а функция `change_s` — операцией смены слайда, который вызывается из библиотеки слайдов My.

Сразу после выполнения выражения `(slide)` диалог загружается в память и инициализируются элементы диалогового окна. В нашем примере к операциям инициализации относятся: загрузка списка изображений в правой части окна (с отметкой верхнего элемента списка) и загрузка первого слайда в левой части окна.

Формирование списка изображений сделано прямо в функции `slide`, а сам список сохранен в рабочей переменной `nlist`. Имена изображений загружаются в элемент `list_box` с помощью функций `start_list`, `add_list` и `end_list`. Выделение в списке первого элемента осуществляется с помощью выражения `(set_tile "names" "0")`, поскольку первый элемент в списке имеет номер 0, а не 1.

Загрузка в левой части диалогового окна первого слайда невозможна без предварительного вычисления размеров элемента `image` с кодом `"sld1"`. Поэтому выражение `(setq dx (dimx_tile "sld1") dy (dimy_tile "sld1"))` позволяет получить размер картинки по горизонтали (`dx`) и вертикали (`dy`) в тех же единицах, которые потребуются в функции загрузки слайда `slide_image`. Только после этого применяются функции `start_image`, `slide_image` и `end_image`, которые непосредственно отображают в левой части окна слайд с именем My(Name1) — для системы такое имя означает, что слайд Name1 необходимо вызвать из библиотеки My (имя файла — My.lib).

По окончании инициализации диалогового окна функция `change_s` обрабатывает все операции пользователя со списком в правой части — как только в списке вы-

брана другая строка, так в левой части отображается слайд с отмеченным именем.

Эта функция вызывается с помощью выражения в кавычках, указанного во втором аргументе функции `action_tile`: `(action_tile "names" "(change_s $value all dx dy)")`. Функции `change_s` в данном выражении передаются следующие аргументы:

- `$value` — номер отмеченного элемента в строковом виде (`"0"`, `"1"`, `"2"` и т. д.);
- `all` — список со всеми именами изображений;
- `dx` — размер зоны слайда в окне по горизонтали;
- `dy` — размер зоны слайда в окне по вертикали.

В теле функции `change_s` сначала получается номер изображения в виде целого числа `i` и имя изображения `name` как элемент, находящийся в полном списке имен на месте с номером `i`.

Затем зона элемента `image` с кодом `"sld1"` заполняется изображением. Для этого между вызовами функций `start_image` и `end_image` находятся вызовы функций `fill_image` (заполняет цветом фона весь прямоугольник) и `slide_image` (прорисовывает слайд). Если прямоугольник не очищать с помощью функции `fill_image`, то слайды будут накладываться друг на друга.

На рис. 3.23 показано то же диалоговое окно, но с другим слайдом, имя которого отмечено в правой части (Name6).

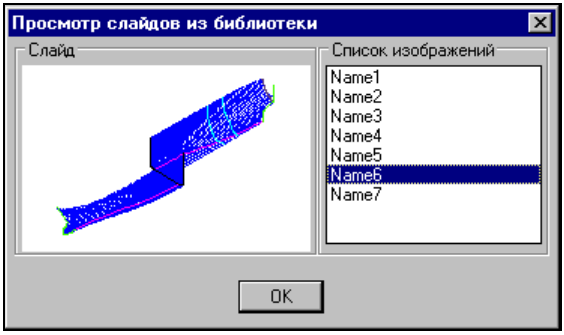


Рис. 3.23. Смена слайда

Напомним этапы формирования библиотеки My.lib со слайдами:

1. С помощью команды MSLIDE (ДСЛАЙД) поочередно создать файлы слайдов Name1.sld, Name2.sld, ... , Name7.sld.
2. Создать текстовый файл (на сопроводительном компакт-диске он называется all.txt) с именами слайдов (Name1, Name2, ...) по одному в каждой строке.
3. С помощью исполняемого файла slidelib.exe (он находится, например, в главной папке программного обеспечения системы AutoCAD и его надо перепи-

сать в ту же папку, где расположены файлы слайдов) создать библиотеку My.lib, для чего в командной строке Windows выполнить

```
slidelib My<all.txt
```

Если все сделать правильно, то в нашем примере будет создан файл библиотеки My.lib с семью слайдами.

3.6.8.2. Изменение цветов графической кнопки

Следующий пример демонстрирует случай, когда в диалоговом окне имеется прямоугольная кнопка, закрашенная некоторым цветом, а пользователю предоставляется возможность независимо изменять цвета левой и правой частей этой кнопки. Для изменения вызывается стандартное окно системы AutoCAD задания цвета.

Пример также является иллюстрацией к анализу координат точки щелчка на кнопке и использования переменной действия `$x`.

В листинге 3.23 размещено описание диалога с графической кнопкой и поясняющим текстом над ней.

Листинг 3.23. Диалог с изменяемым цветом графической кнопки

```
// Глава 03\Book06\imcolor.dcl
imcol: dialog{label="Смена цветов графической кнопки";
:column{label="Щелкни";
  :text{label=" по той половине кнопки,";}
  :text{label=" в которой надо изменить цвет";}
  :image_button{key="col2";color=2;width=24;height=6.5;}
}//column
spacer_1;
ok_button;
}//imcol
```

В начальный момент кнопка имеет желтый цвет, единый для обеих частей (левой и правой). На рис. 3.24 показано состояние диалогового окна в тот момент, когда двум частям кнопки назначены разные значения цвета.

Текст LISP-функций, работающих с диалогом `imcol`, приведен в листинге 3.24.

Листинг 3.24. Функции, работающие с диалогом `imcol`

```
; Глава 03\Book06\imcolor.lsp
; Функция загрузки диалога
(defun imcolor ( / dcl_id dx dy)
(setq dcl_id (load_dialog "imcolor.dcl"))
(new_dialog "imcol" dcl_id)
```

```
; Сохраним цвет кнопки, заданный в DCL-файле
(setq g_c1 (get_attr "col2" "color") g_c2 g_c1)
; Размеры графической кнопки
(setq dx (dimx_tile "col2") dy (dimy_tile "col2"))
; Обработка щелчка по кнопке
(action_tile "col2" "(change_c dx dy)")
(start_dialog)
(unload_dialog dcl_id)
); defun imcolor
; Функция смены цвета частей кнопки
(defun change_c (x y / c nc xm)
  (setq xm (/ x 2)); абсцисса середины кнопки
  (setq c (if (< $x xm) g_c1 g_c2))
  ; Вызов диалога выбора цвета
  (setq nc (cdr (assoc 62 (acad_truecolordlg (cons 62 (atoi c))))))
  (if nc
    (progn
      ; Заполнение части прямоугольника новым цветом
      (start_image "col2")
      (if (< $x xm)
        (progn
          (fill_image 0 0 xm y nc)
          (setq g_c1 (itoa nc))
        ); progn
        (progn
          (fill_image xm 0 x y nc)
          (setq g_c2 (itoa nc))
        ); progn
      ); if
      (end_image)
    ); progn
  ); if
); defun change_c
```

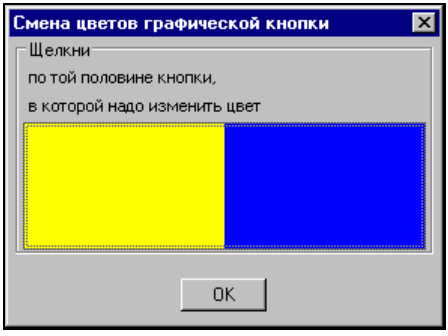


Рис. 3.24. Пример изменения цветов графической кнопки

Первая функция `imcolor` загружает диалоговое окно и заполняет всю кнопку желтым цветом. Значения цветов для левой и правой частей кнопки сохраняются в глобальных переменных `g_c1` и `g_c2` (на стадии инициализации им присваивается значение 2).

Вторая функция `change_c` вызывается в тот момент, когда пользователь щелкнет внутри прямоугольника графической кнопки. Координаты точки щелчка сохраняются системой в двух глобальных переменных действия: `$x` и `$y`. Для нужд примера нам достаточно анализировать только первую переменную, поскольку ее значения достаточно для того, чтобы понять, выполнен щелчок в левой или в правой части кнопки.

Сразу после щелчка в зоне кнопки с помощью функции `acad_truecolordlg` вызывается стандартное диалоговое окно **Select Color** (Выбор цвета), которое имеет три вкладки: **Index Color** (Номер цвета), **True Color** (Вся палитра) и **Color Books** (Альбомы цветов). Пользователь имеет возможность задать новое значение цвета на любой из этих вкладок.

Однако язык DCL поддерживает только основные цвета с номерами от 0 до 256. Поэтому при выборе, например, цвета с помощью RGB-составляющих на вкладке **True Color** (Вся палитра) необходим метод, по которому этому цвету подбирается ближайший из требуемого интервала значений номеров. Такую возможность предоставляет примененная в листинге 3.24 функция `acad_truecolordlg`. Напомним, что в возвращаемом значении функции всегда присутствует точечная пара с DXF-кодом 62, которая и содержит нужный нам номер цвета. Например: `((62 . 15) (420 . 11285051) (430 . "RAL CLASSIC$RAL 3031")).`

После выбора нового номера цвета необходимо с помощью функции `fill_image` закрасить нужную половину кнопки этим цветом. Для выбора левой или правой половины кнопки достаточно сравнить значение переменной `$x` с абсциссой середины (`xm`).

Дальнейшее понятно. Надо только не забыть обновить значение одной из глобальных переменных, хранящих текущие номера цветов.

3.6.8.3. Изменение рисунка, наносимого на кнопку

Третий пример диалога с графическими элементами касается использования функции `vector_image`, рисующей внутри элемента отрезки заданного цвета. Текст DCL-файла приведен в листинге 3.25.

Листинг 3.25. Диалог с возможностью рисования отрезков на кнопке

```
// Глава 03\Book07\imvector.dcl
imvec: dialog{label="Рисование на графической кнопке";
:row{
  :column{label="Зона рисования";
```

```
    :image_button{key="im";color=0;width=50;height=8.5;}
  }//column
  spacer_1;
  :column{children_alignment=right;
          children_fixed_width=true;
    :row{
      :text{label="Цвет";}
      :image_button{color=7;width=5;height=3;key="col";}
    }//row
    :edit_box{label="Точка 1";edit_width=3;key="x1";}
    :edit_box{label="";edit_width=3;key="y1";}
    :edit_box{label="Точка 2";edit_width=3;key="x2";}
    :edit_box{label="";edit_width=3;key="y2";}
  }//column
}//row
spacer_1;
ok_button;
}//imvec
```

Начальное состояние этого диалогового окна показано на рис. 3.25.

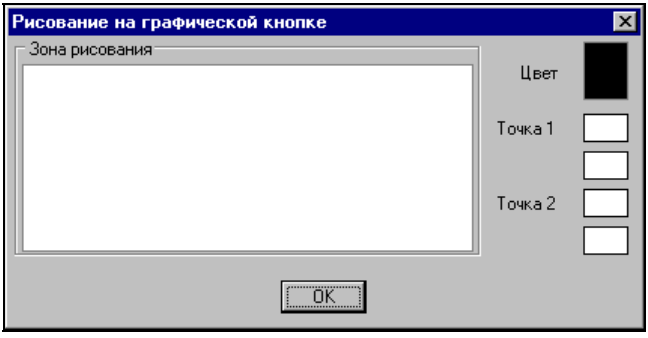


Рис. 3.25. Начальное состояние диалогового окна **Рисование на графической кнопке**

В левой части расположена зона, в которой можно щелчками мыши рисовать отрезки. Цвет создаваемого отрезка указывается графической кнопкой **Цвет**. Если цвет необходимо изменить, то следует щелкнуть по этой кнопке и в открывшемся стандартном окне выбора цвета задать новый цвет.

Для рисования отрезка необходимо два раза щелкнуть в зоне рисования, указав таким образом две конечные точки отрезка, координаты которых выводятся в текстовые поля **Точка 1** и **Точка 2**. Программа нарисует отрезок с концами в этих точках и значением цвета, отображаемым кнопкой **Цвет**. Рабочее состояние окна показано на рис. 3.26.

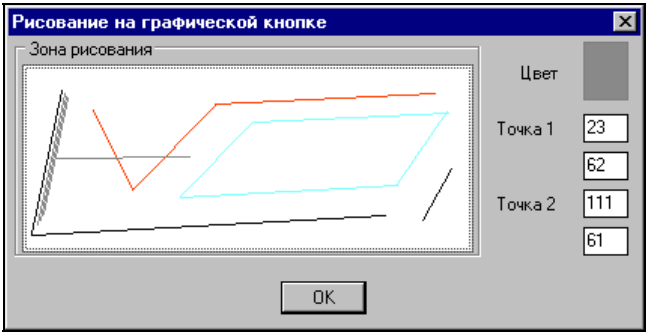


Рис. 3.26. Рабочее состояние диалогового окна **Рисование на графической кнопке**

Для реализации такого поведения диалогового окна написаны функции, приведенные в листинге 3.26.

Листинг 3.26. Функции рисования на кнопке

```
; Глава 03\Book07\imvector.lsp
; Функция загрузки диалога
(defun imvector ( / dcl_id dx dy)
(setq dcl_id (load_dialog "imvector.dcl"))
(new_dialog "imvec" dcl_id)
; Цвет рисования
(setq g_c "7")
; Номер следующей точки
(setq g_np 1)
; Координаты точек
(setq g_x1 nil g_y1 nil g_x2 nil g_y2 nil)
; Размеры графической кнопки (зоны рисования)
(setq dx (dimx_tile "im") dy (dimy_tile "im"))
; Обработка щелчка по кнопке Цвет
(action_tile "col" "(sel_c dx dy)")
; Обработка щелчков в зоне рисования
(action_tile "im" "(drawb)")
(start_dialog)
(unload_dialog dcl_id)
); defun imvector
; Функция выбора цвета рисования
(defun sel_c (x y / nc)
  ; Вызов диалога выбора цвета
  (setq nc (cdr (assoc 62 (acad_truecolordlg (cons 62 (atoi g_c))))))
  (if nc
    (progn
      (setq g_c (itoa nc))
```

```
      (start_image "col")
      (fill_image 0 0 x y nc)
      (end_image)
    ); progn
  ); if
); defun sel_c
; Функция обработки точки
(defun drawb ( / )
  (setq g_np (1+ (- 2 g_np)))
  (if (= g_np 1)
    (progn
      (setq g_x2 $x g_y2 $y)
      (start_image "im")
      (vector_image g_x1 g_y1 g_x2 g_y2 (atoi g_c))
      (end_image)
      (set_tile "x2" (itoa $x))
      (set_tile "y2" (itoa $y))
    ); progn
    (progn
      (setq g_x1 $x g_y1 $y)
      (set_tile "x1" (itoa $x))
      (set_tile "y1" (itoa $y))
      (set_tile "x2" "")
      (set_tile "y2" "")
    ); progn
  ); if
); defun drawb
```

В данном листинге три функции. Функция `imvector` управляет загрузкой диалогового окна. На стадии инициализации окна в этой функции задаются глобальные переменные `g_c` (текущий цвет, в строковом виде), `g_np` (номер следующей точки, 1 или 2), `g_x1`, `g_y1`, `g_x2`, `g_y2` (координаты первой и второй точек), а также вычисляются локальные переменные `dx` и `dy` (размеры зоны рисования).

В случае щелчка внутри графической кнопки с кодом `"col"` вызывается вторая функция (`sel_c`), а в случае щелчка внутри графической кнопки с кодом `"im"` (зона рисования) — третья функция (`drawb`).

Функция `sel_c` имеет примерно ту же структуру, что и функция `change_c` в предыдущем примере (см. листинг 3.24). Она позволяет выбрать цвет с помощью стандартного окна **Select Color** (Выбор цвета).

Функция `drawb` осуществляет считывание координат точек щелчков (поочередно первая — вторая) и рисования отрезков после задания второй точки. Кроме того, координаты точек выводятся в текстовые поля справа. Выбор операции (рисовать или еще не рисовать) определяется по текущему значению глобальной переменной `g_np`.

На практике, как правило, функция vector_image используется не для интерактивного рисования, а для формирования простого изображения из отрезков на графическом элементе или получения дополнительных эффектов (перечеркивания слайда и т. п.).

3.7. Дополнительные приемы программирования диалоговых окон

В данном разделе рассмотрены два примера нестандартных вариантов программирования диалоговых окон.

3.7.1. Временный выход из диалогового окна

Текст функции tex_point, иллюстрирующей выход из диалогового окна с последующим возвратом в окно, приведен в листинге 3.27, а текст используемого файла tex.dcl — в листинге 3.28.

Листинг 3.27. Функция tex_point

```
; Глава 03\Book08\tex.lsp
(defun tex_point (/ dcl_id step pt)
  (setq dcl_id (load_dialog "tex.dcl"))
  (setq step 2)
  (if (null pt)(setq pt (list 0.0 0.0 0.0)))
  (while (>= step 2)
    (if (null (new_dialog "texp" dcl_id))(exit))
    (set_tile "xp" (rtos (car pt) 2 5))
    (set_tile "yp" (rtos (cadr pt) 2 5))
    (set_tile "zp" (rtos (caddr pt) 2 5))
    (action_tile "accept"
      (strcat
        "(setq x (atof (get_tile \"xp\")))"
        "(setq y (atof (get_tile \"yp\")))"
        "(setq z (atof (get_tile \"zp\")))"
        "(setq pt (list x y z))"
        "(done_dialog 1)"
     );strcat
   );action_tile
    (setq step (start_dialog))
    (cond
      ((= step 3)(setq pt (getpoint "\nУкажите точку: ")))
      ((= step 1)
        (princ "\nТочка: (")
        (princ (rtos x 2 5))(princ " ")
```

```
        (princ (rtos y 2 5))(princ " ")
        (princ (rtos z 2 5))(princ ")")
     );= step 1
   );cond
 );while
  (unload_dialog dcl_id)
  (princ)
);defun tex_point
```

Листинг 3.28. Файл tex.dcl

```
// Глава 03\Book08\tex.dcl
texp: dialog {label="Задание точки";
:text{label="Введите координаты";}
:text{label="или укажите точку на экране";}
:spacer{height=1;}
:row{
  :column{
    :edit_box{label="X:";value="0";key="xp";}
    :edit_box{label="Y:";value="0";key="yp";}
    :edit_box{label="Z:";value="0";key="zp";}
  }//column
  :button{label="Указать <";key="mousep";height=6;fixed_width=true;
          action="(done_dialog 3)";}
}//row
spacer_1;
ok_button;
}//texp
```

Для выполнения данного примера необходимо загрузить и выполнить функцию `tex_point`. Система AutoCAD откроет диалоговое окно, приведенное на рис. 3.27.

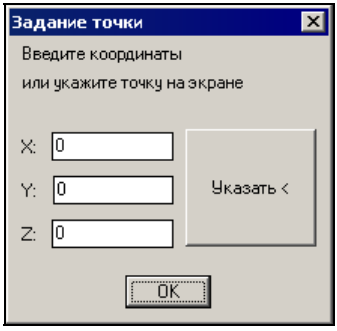


Рис. 3.27. Диалоговое окно **Задание точки**

В первой строке после комментариев файла tex.dcl задаются имя диалога (`texp`) и текст заголовка окна. Вторая и третья строки диалогового окна являются простым текстом, формируемым с помощью элементов типа `text`.

Ниже, после промежутка, заданного элементом `spacer`, располагается группа элементов, оформленная в виде строки (оператор `row`), которая является объединением из колонки (оператор `column`) и кнопки (элемент `button`). Колонка состоит из трех вертикально расположенных полей ввода текста (поля `edit_box`). Кнопка имеет надпись `"Указать <"` и используется для временного выхода из окна при указании точки с помощью мыши.

Далее в окне идет промежуток, формируемый элементом `spacer_1`, и кнопка **OK**, закрывающая окно и завершающая работу функции `tex_point`.

Находясь в диалоговом окне, пользователь может либо ввести в полях **X**, **Y** и **Z** три координаты, либо нажать кнопку **Указать**. Во втором случае приложение временно закрывает окно и выдает запрос ***Укажите точку:***. После указания точки приложение вновь открывает окно, причем координаты указанной точки отображаются в полях **X**, **Y** и **Z**. Когда координаты окончательно сформированы (а их можно изменить или указать еще раз), пользователь нажимает кнопку **OK**, и приложение выдает сообщение в следующей форме:

Точка: (457.92577 166.14289 20.39021)

Для достижения эффекта временного выхода из диалогового окна в DCL-файле в атрибуте `action` кнопки с кодом `"mousep"` использована функция `done_dialog` с аргументом `3`, который автоматически становится возвращаемым значением функции `start_dialog`. Поэтому переменная `step` получает значение `3`, что обрабатывается программой и заставляет снова открыть диалоговое окно после указания точки мышью.

3.7.2. Вложенные диалоговые окна

Возможна ситуация, когда по нажатию какой-то кнопки текущего диалогового окна нужно открыть следующее окно, из которого потом необходимо вернуться в предыдущее. Поэтому возникает необходимость работать с вложенными друг в друга диалоговыми окнами.

Система AutoCAD рассчитана на поддержку восьми уровней вложенности диалоговых окон, но реально рекомендуется использовать не более четырех уровней (кроме того, много одновременно открытых окон неудобно для пользователя).

Программирование вложенных окон достаточно просто — можно в LISP-выражение, используемое в качестве второго аргумента функции `action_tile`, ввести обращение к функциям `new_dialog` и `start_dialog`. Можно посоветовать в таких случаях в одном DCL-файле разместить описания всех диалогов (основно-

го и вложенных), чтобы обращение к функции `load_dialog` выполнить только один раз и не перегружать LISP-выражения в функции `action_tile`.

В листинге 3.29 приведен текст файла twodia.dcl, содержащего описание двух простых диалогов (`dia1` и `dia2`). Диалог `dia2` вызывается из диалога `dia1` нажатием кнопки **Вызвать следующее окно**.

Листинг 3.29. Пример вложенных диалоговых окон

```
// Глава 03\Book09\twodia.dcl
dia1: dialog{label="Диалоговое окно первого уровня";
:edit_box{label="Ввод названия:";key="t1";}
:button{label="Вызвать следующее окно";key="b1";}
ok_button;
}// dia1
dia2: dialog{label="Диалоговое окно второго уровня";
:text{label="Это вложенное окно";key="t2";}
:ok_button{label="Вернуться";}
}// dia2
```

В листинге 3.30 приведен текст файла twodia.lsp, в котором находится текст двух функций. Функция `twodia` открывает первое диалоговое окно и при нажатии кнопки **Вызвать следующее окно** обращается к функции `opendia2`, открывающей вложенное диалоговое окно.

Листинг 3.30. Файл twodia.lsp

```
; Глава 03\Book09\twodia.lsp
(defun twodia ( / dcl_id)
  (if (< (setq dcl_id (load_dialog "twodia.dcl")) 0)(exit))
  (if (not (new_dialog "dia1" dcl_id)) (exit))
  (action_tile "t1" "(setq mytxt $value)")
  (action_tile "b1" "(opendia2 dcl_id)")
  (start_dialog)
  (unload_dialog dcl_id)
  (princ)
);defun twodia
(defun opendia2 (dc /)
  (if (not (new_dialog "dia2" dc)) (exit))
  (start_dialog)
);defun opendia2
```

На рис. 3.28 приведен вид диалоговых окон, открытых с помощью функций `twodia` и `opendia2` (для удобства первое окно смещено, чтобы вложенное в него окно не закрыло первое полностью).

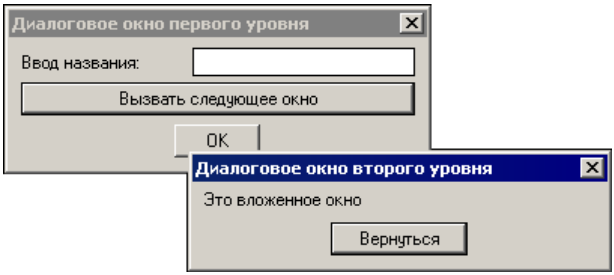


Рис. 3.28. Вложенные диалоговые окна

3.7.3. Применение паролей

При работе с полем `edit_box` бывает необходимость шифровать текст, вводимый пользователем, чтобы он не был виден (например, если вводится пароль). Для обеспечения нужного эффекта следует использовать атрибут `password_char`. Символ, заданный в качестве значения этого атрибута, заменяет в диалоговом окне символы вводимого пользователем текста. Программа же, которая читает значение этого поля, получит текст в незашифрованном виде (например, для проверки правильности введенного пароля).

В листинге 3.31 приведен пример простого диалога, в котором поле `edit_box` должно быть зашифровано.

Листинг 3.31. Пример диалога с шифрованным полем

```
// Глава 03\Book10\pass.dcl
pass: dialog{label="Диалоговое окно с паролем";
:edit_box{label="Введите пароль:";key="e1";password_char="*";}
ok_button;
}// pass
```

На рис. 3.29 приведено диалоговое окно, рассмотренное в листинге 3.31. В этом окне текст, вводимый в поле `edit_box`, закрыт звездочками.

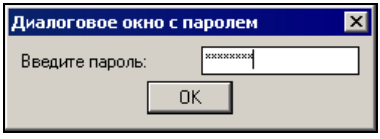


Рис. 3.29. Диалоговое окно с зашифрованным паролем

ГЛАВА 4

Меню и адаптации

Такие элементы пользовательского интерфейса системы AutoCAD, как падающие меню, экранные меню, панели инструментов, контекстные меню, образуют управляемое объединение. Это объединение в версиях системы до 2005 включительно называлось *меню*. Начиная с версии 2006 объединение стало называться *адаптацией*.

В данной главе рассматриваются вопросы взаимодействия пользователя через LISP-программы с меню, адаптациями и их элементами (пунктами, кнопками и т. д.).

4.1. Коротко о меню и адаптациях

Меню (в версиях до 2005) порождается файлами с четырьмя расширениями:

- mnu — файл шаблона меню;
- mnc — файл компилированного меню;
- mnr — файл ресурсов меню (содержит пиктограммы кнопок панелей инструментов);
- mns — файл текстовой части меню.

Примером таких файлов являются файлы acad.mnu, acad.mnc, acad.mnr и acad.mns, расположенные в папке Support текущего пользователя системы AutoCAD.

В системе AutoCAD 2006 адаптации порождаются файлами с двумя расширениями:

- cui — файл адаптации;
- mnr — файл ресурсов адаптации (содержит пиктограммы кнопок панелей инструментов);

Соответствующие файлы базовой адаптации — acad.cui и acad.mnr, которые тоже размещаются в папке Support текущего пользователя системы AutoCAD.

Даже если вы работаете только с меню acad системы AutoCAD, то оно все время меняется. В сеансе редактирования вы модифицируете состояние, например, панелей инструментов, их количество и положение.

Подробнее о меню и адаптациях можно познакомиться в книге "AutoCAD: разработка приложений, настройка и адаптация"[1].

4.1.1. Группы меню и группы адаптации

В текущих меню и адаптациях обязательно есть независимые части, называемые *группами меню* и *группами адаптации*. Каждая группа соответствует одному шаблону меню (файлу с расширением mnu) или одной адаптации (файлу с расширением cui). Стандартное меню системы AutoCAD имеет группу с именем ACAD, которую можно назвать *базовой группой*.

Текущий пользовательский интерфейс может быть дополнен падающими, экранными и другими меню, панелями инструментов из других групп, называемых в версиях до 2005 *фрагментными меню* или с версии 2006 *частичными адаптациями* (например, падающее меню и панели Express Tools относятся к группе EXPRESS). Система AutoCAD предоставляет средства для загрузки и выгрузки групп меню (адаптации), добавления и удаления меню в строку меню, добавления, редактирования и удаления пунктов меню, кнопок панелей и т. д. (часть операций реализуются только с применением технологии ActiveX).

4.1.2. Разделы меню (адаптации)

Внутри шаблона меню или адаптации каждый вид элементов интерфейса (падающие меню, контекстные меню и т. д.) делится на следующие разделы:

- BUTTONS — кнопочные меню;
- AUX — дополнительные кнопочные меню;
- POP — падающие и контекстные меню;
- TOOLBARS — панели инструментов;
- IMAGE — графические меню со слайдами;
- SCREEN — экранные меню;
- TABLET — планшетное меню;
- HELPSTRINGS — строки подсказок;
- ACCELERATORS — горячие клавиши.

---

[1] Полещук Н. Н. AutoCAD: разработка приложений, настройка и адаптация. — СПб.: БХВ-Петербург, 2006. — 992 с. (+CD-ROM).

Разделы BUTTONS, AUX, POP и TABLET имеют еще в конце названия номер (например, BUTTONS2, POP0, POP12, AUX1), в то время как остальные разделы номеров не содержат.

4.2. Структура заголовков и пунктов меню

Заголовок меню имеет такую структуру:

```
<тег>   [<заголовок>]
```

Здесь `<тег>` — это тег (внутреннее имя, tag), используемый для работы программного обеспечения системы с заголовком в оперативной памяти; `<заголовок>` — это текст, который выводится в качестве заголовка меню. Тег обычно начинается с символов `ID_`.

Пункт меню имеет похожую структуру:

```
<тег>   [<текст>]<макрос>
```

Здесь `<тег>` — это тег, обычно начинающийся с символов `ID_` и отличающийся от тегов других пунктов и заголовков меню; `<текст>` — это текст, являющийся наименованием пункта меню (выводится на экран); `<макрос>` — текстовый макрос, содержащий операции, которые выполняет пункт меню.

Пункты некоторых разделов не имеют тегов — это разделы BUTTONS, AUX и IMAGE. В разделах SCREEN и TABLET теги могут присутствовать, но они системой AutoCAD игнорируются.

Наименование пункта меню может программно меняться. В нем, например, может появляться символ отметки (флажок). Наименование пункта может гаситься, если пункт отключается пользовательской программой.

В макрос могут входить команды системы AutoCAD и их опции, выражения LISP и языка DIESEL, а также специальные комбинации символов, имитирующих, например, нажатие клавиш.

Особые пункты меню — это *разделители*. Они не имеют наименования и макроса.

4.3. Функции LISP, работающие с меню

В языке AutoLISP имеются две функции, позволяющие выполнять над элементами падающих, графических и экранных меню некоторые операции: отключать, изменять, удалять и т. д.

4.3.1. Функция *menugroup*

Проверяет, загружена ли группа меню (адаптации).

```
(menugroup <группа>)
```

Аргумент — текстовая строка с именем группы.

Возвращаемое значение — строка с именем группы, если она загружена; иначе возвращается `nil`.

Примеры:

- `(menugroup "ACAD")` возвращает `"ACAD"`
- `(menugroup "EXPRESS")` возвращает `nil`, если группа EXPRESS еще не загружена

4.3.2. Функция *menucmd*

Выполняет операции над элементами меню.

```
(menucmd <строка>)
```

Аргумент — строка специального вида, в которой указано, какая и с каким элементом меню выполняется операция (см. ниже).

Возвращаемое значение — `nil`.

С помощью аргумента `<строка>` функция `menucmd` может открыть падающее меню, погасить пункт меню, включить пункт меню и т. п. Значение аргумента должно иметь следующий вид:

```
"<зона>=<действие>"
```

Левая часть аргумента `<строка>` (параметр `<зона>`) отделяется от правой (параметра `<действие>`) знаком равенства. Параметр `<зона>` должен принимать одно из значений, приведенных в табл. 4.1.

Таблица 4.1. *Допустимые значения параметра* `<зона>`

Значение	**Описание**
B1—B4	Соответствует разделам BUTTONS1—BUTTONS4
A1—A4	Соответствует разделам AUX1—AUX4
P0—P25	Соответствует разделам POP0—POP25
I	Соответствует разделу IMAGE
S	Соответствует разделу SCREEN
T1—T4	Соответствует разделам TABLET1—TABLET4
M	Вводится выражение на языке DIESEL
G*группа.имя*	Соответствует разделу с именем *имя* группы с наименованием *группа* (например, "GACAD.POP1")

В качестве значений параметра `<действие>` может использоваться либо имя раздела (или пункта) меню, либо символы из табл. 4.2.

Таблица 4.2. *Допустимые значения параметра* `<действие>`

Символы	Описание
*	Активизация
+	Добавление к строке меню
-	Удаление
!.	Установка флажка около наименования пункта
~	Отключение (гашение) пункта
?	Справка о состоянии пункта
#?	Расширенная справка
|	Выполнение макроса пункта меню
(пустое значение)	Восстановление состояния пункта

Подробнее операции с разными разделами и пунктами меню будут рассмотрены в данной главе ниже.

4.3.3. Операции загрузки (выгрузки) групп

Для операций загрузки групп меню из MNU- или CUI-файлов и выгрузки групп из памяти специальных функций нет, поэтому необходимо использовать для этих целей команды системы AutoCAD, например:

- `(command "_.MENU" "d:\\My\\3menu")` — загрузка группы базового меню или базовой адаптации
- `(command "_.MENULOAD" "d:\\My\\3menu")` — загрузка группы фрагментного меню или частичной адаптации
- `(command "_.MENUUNLOAD" "3menu")` — выгрузка группы фрагментного меню или частичной адаптации
- `(command "_.CUILOAD" "d:\\My\\3menu")` — в версии 2006 загрузка группы частичной адаптации
- `(command "_.CUIUNLOAD" "3menu")` — в версии 2006 выгрузка группы частичной адаптации

4.4. Падающие меню

Падающим меню в каждой группе отведены разделы от POP1 до POP499. В разных группах могут быть разделы с одинаковыми именами (например, раздел POP1 есть как в группе ACAD, так и в группе EXPRESS).

При этом в строке падающих меню одновременно может быть до 25 заголовков из числа загруженных в память групп меню (адаптации). В данном разделе книги рассматриваются вопросы программирования различных операций с падающими меню.

4.4.1. Добавление в строку меню

Для добавления заголовка в строку меню программными средствами следует использовать следующее LISP-выражение:

```
(menucmd "P10=+EXPRESS.POP1")
```

Это выражение добавляет заголовок **Express** (имя меню, расположенное в разделе POP1 группы EXPRESS) к строке падающих меню на десятое место, если группа EXPRESS уже загружена в память.

Вместо имени раздела (POP1) можно использовать имя меню, которое следует в файле шаблона меню с префиксом из двух звездочек за строкой с именем раздела или в файле адаптации описаны как `Alias`. Например:

```
(menucmd "P10=+ACAD.HELP")
```

Если вам нужно расположить новый заголовок не на конкретном по номеру месте в строке падающих меню, а перед ранее добавленным заголовком с определенным именем, можно воспользоваться таким выражением:

```
(menucmd "GACAD.POP12=+EXPRESS.POP1")
```

В этом случае заголовок из раздела POP1 группы EXPRESS будет добавлен в строку падающих меню перед заголовком меню, импортированного из раздела POP12 группы меню ACAD, независимо от того, на каком месте находится последний заголовок (а он может оказаться, например, первым или седьмым).

4.4.2. Удаление из строки меню

Удаление заголовка из строки падающих меню может быть выполнено с помощью следующего LISP-выражения:

```
(menucmd "P10=-")
```

Данное выражение удалит падающее меню, заголовок которого находится на десятом месте в строке падающих меню.

Если нужно удалить меню, импортированное из раздела POP1 группы EXPRESS, независимо от места его расположения, следует применить такое выражение:

```
(menucmd "GEXPRESS.POP1=-")
```

4.4.3. Вывод содержимого меню на экран

Для того чтобы вывести на экран меню, заголовок которого является пятым в строке падающих меню, нужно выполнить следующее выражение:

```
(menucmd "P5=*")
```

Результат выполнения показан на рис. 4.1.

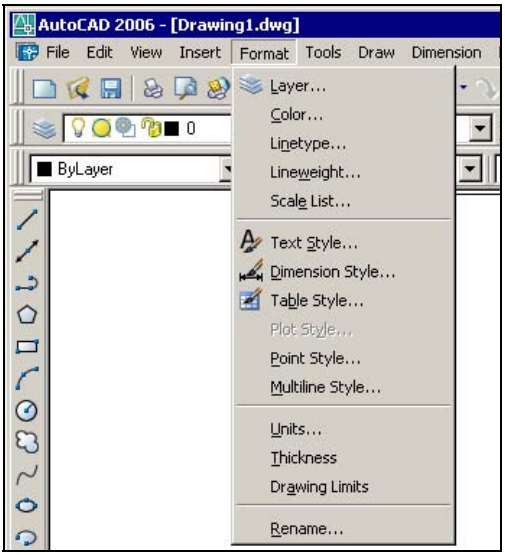


Рис. 4.1. Вывод содержимого падающего меню

Можно обратиться к меню по имени группы и имени раздела, из которого это меню было загружено в память:

```
(menucmd "GACAD.POP5=*")
```

Возможен вариант обращения к меню не по имени раздела, а по имени меню:

```
(menucmd "GACAD.FORMAT=*")
```

Эти операции раскрывают меню **Format**, которое находится в разделе POP5 файла acad.mnu или acad.cui.

4.4.4. Отметка пункта меню

В операциях, выполняемых с помощью функции `menucmd`, обращение к пункту меню осуществляется с помощью номера, вводимого в параметре `<зона>` после символов, обозначающих раздел (P5, P12 и т. п.), через точку. Номер 0 соответствует заголовку меню. Разделительные строки в падающих меню тоже участвуют в нумерации пунктов.

У наименования пункта падающего меню может стоять отметка в виде флажка (галочки). Обычно это бывает удобно, когда пункт меню отражает настройку, для которой нужно показать, что она включена. Если в файле шаблона наименование пункта начинается с символов "!.", то эти два символа при выводе наименования пункта на экран преобразуются в знак отметки.

Следующее выражение ставит слева от наименования третьего пункта десятого меню знак отметки:

```
(menucmd "P10.3=!.")
```

На рис. 4.2 приведен пример меню с отмеченными пунктами.

Рис. 4.2. Отметка пункта меню

4.4.5. Отключение пункта меню

Символ "~" перед наименованием пункта меню делает этот пункт недоступным (приглушает). Пример:

```
(menucmd "P10.3=~")
```

Это выражение отключает третий сверху пункт меню, заголовок которого в данный момент является десятым в строке падающих меню.

Отметка пункта меню не препятствует его отключению.

Программное обеспечение системы AutoCAD контролирует состояние большинства пунктов группы ACAD и не дает возможности отключать их пользовательским программам.

4.4.6. Восстановление пункта меню

Для того чтобы восстановить пункт падающего меню (снять отключение), необходимо выполнить следующую операцию:

```
(menucmd "P10.3=")
```

Данное действие восстановит пункт с номером три из десятого меню.

4.4.7. Замена меню

Возможна замена падающего меню на другое, которое еще не фигурирует в пользовательском интерфейсе. Схема замены следующая:

```
(menucmd "P10=RITM.POP13")
```

Данная операция заменит десятое падающее меню в пользовательском интерфейсе на меню из раздела POP13 группы RITM.

Можно при замене опускать имя группы, если вместо предыдущего меню вставляется меню из группы базового меню, например:

```
(menucmd "P10=POP12")
```

Если в качестве базового в данный момент выбрано стандартное меню системы AutoCAD, то в этом случае десятое меню заменится на меню из раздела POP12 группы ACAD.

Вместо имени раздела (POP13, POP12) можно было использовать имена меню, которые в файле шаблона следуют с префиксом из двух звездочек за строкой с именем раздела или в файле адаптации описаны как `Alias`.

Примеры:

- `(menucmd "P21=RITM.POP20")`
- `(menucmd "P10=HELP")`

4.4.8. Справка о состоянии пункта меню

Справку о состоянии пункта можно получить с помощью функции `menucmd`, используя знак вопроса в качестве значения параметра `<действие>`. Пример:

```
(menucmd "P11.1=?")
```

Это выражение выводит состояние первого пункта одиннадцатого падающего меню.

Возвращается `"~"`, если пункт 1 одиннадцатого падающего меню отключен; `"!"`, если пункт содержит отметку слева от наименования; `""`, если пункт находится в обычном состоянии (включен и без отметки); `nil`, если нет падающего меню с таким номером.

Если пункт и отключен и отмечен, то возвращается `"~!"` или `"!~"` в зависимости от порядка действий, выполненных над пунктом.

Возможно выполнение операции расширенной справки, для чего перед знаком вопроса следует добавлять символ `"#"`:

```
(menucmd "P11.1=#?")
```

Это выражение возвращает состояние пункта, но с префиксом, например: `"P11.1="` или `"P11.1=!"`.

4.5. Экранные меню

Раздел с именем SCREEN шаблона меню содержит экранные меню, которые в ранних версиях AutoCAD были основным видом меню, а в последней версии

применяется лишь частью пользователей (хотя нельзя сказать, что такие меню себя полностью исчерпали).

Смена экранного меню выполняется с помощью вызова функции `menucmd`, в котором параметр `<зона>` равен `"S"` , например:

♦ `(menucmd "S=ACAD.OSNAP")` загружает меню OSNAP в зону экранных меню

4.6. Графические меню

Графические меню используют заранее подготовленные слайды и библиотеки слайдов. Описание графических меню располагается в разделе IMAGE.

При вызове графического меню следует иметь в виду особенность: графическое меню надо сначала активизировать, а затем вывести с помощью звездочки.

Пример вызова меню IMAGE_VPORTI группы ACAD из программы на языке AutoLISP:

♦ `(menucmd "I=IMAGE_VPORTI")`

`(menucmd "I=*")`

На рис. 4.3 показано окно, выводимое графическим меню IMAGE_VPORTI.

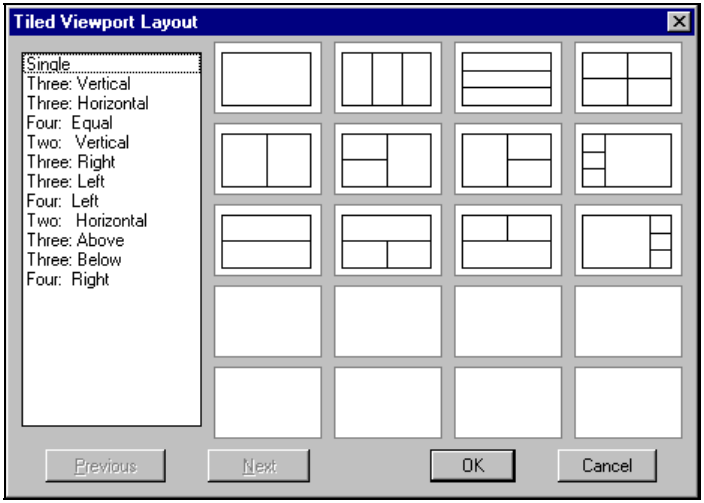


Рис. 4.3. Вызов графического меню

4.7. Панели инструментов

Раздел TOOLBARS является одним из самых больших разделов меню и содержит информацию о панелях инструментов, их структуре и начальном располо-

жении. Работа с панелями возможна только с применением средств ActiveX *(см. приложение 5)*.

4.8. Контекстные меню

Контекстные меню — это меню, которые вызываются по нажатии правой кнопки мыши и пункты которых зависят от контекста (например, от имени выполняемой команды).

4.8.1. Разделы с контекстными меню

Контекстным меню в файле шаблона соответствуют разделы POP0, POP500, POP501 и другие разделы с именами типа POP*n*, где *n* принимает значения от 500 до 999. Меню POP0 вызывается щелчком правой кнопки мыши при нажатой клавише <Shift> (фактически оно вызывается через меню AUX2), остальные — щелчком правой кнопки мыши без клавиши <Shift>.

Структура разделов контекстных меню аналогична структуре разделов падающих меню.

4.8.2. Схема работы контекстных меню

Контекстное меню GRIPS вызывается по щелчку правой кнопки мыши, если в рисунке высвечены ручки объектов, и одна из ручек выбрана (т. е. имеет красный цвет).

Меню CMDEFAULT работает в тот момент, когда нет высвеченных ручек, и не выполняется никакая команда системы AutoCAD.

Если нет работающей команды, но высвечены ручки и ни одна из них не выбрана, то вызывается меню CMEDIT. К этому меню добавляется содержимое меню, имя которого формируется по следующему принципу. Если выбран только один объект, то меню должно иметь префикс `OBJECT_`. Если выбрано несколько объектов, то меню должно иметь префикс `OBJECTS_`. Далее к имени нужно добавить тип выбранного объекта или объектов (если выбрано несколько объектов, то они должны быть одного типа). Поэтому к зарезервированным относятся такие имена меню, как OBJECT_TEXT, OBJECT_LINE, OBJECTS_DIMENSION и т. п. На рис. 4.4 приведено контекстное меню, вызываемое при выделении с помощью ручек объекта типа LWPOLYLINE.

Содержимое этого меню сложилось из содержимого меню CMEDIT и меню OBJECT_LWPOLYLINE. Из последнего меню был взят пункт **Edit Polyline** (Редактирование полилинии).

Контекстное меню с именем CMCOMMAND работает, когда нет выделенных объектов, и выполняется команда системы AutoCAD или команда, определенная пользовательским приложением. К пунктам этого меню в качестве строк добав-

ляются опции, которые выведены в командную строку в квадратных скобках и разделены символами "/". На рис. 4.5 приведен пример контекстного меню, появляющегося во время работы команды ПОЛРЕД (PEDIT).

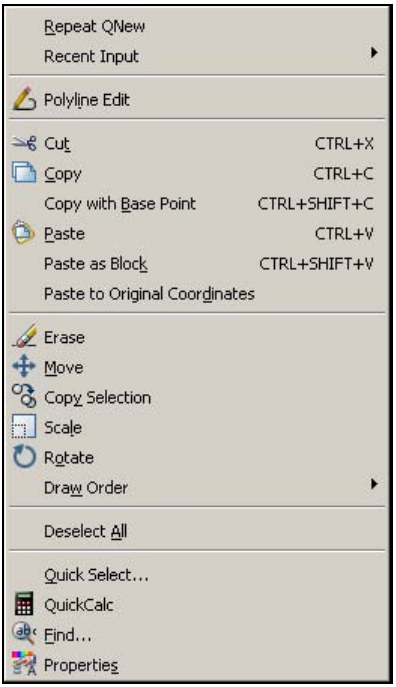


Рис. 4.4. Контекстное меню редактирования компактной полилинии

К пунктам контекстного меню CMCOMMAND при выводе добавляются еще пункты тех контекстных меню, которые имеют имя, начинающееся с префикса COMMAND_, а далее присоединено имя выполняемой команды. Так, например, при выполнении команды ABCD будет проверено наличие меню с именем COMMAND_ABCD, и если такое контекстное меню существует, то его пункты будут добавлены к другим выводимым пунктам.

При программировании приложений необходимо учитывать возможности контекстных меню и при выводе опций в командную строку использовать квадратные скобки и разделяющие символы наклонной черты.

Пример:

```
(setq pt1 (getpoint "Основная точка объекта или [A/B/H]: "))
```

Если выполнить это выражение, то в командной строке появится текст, заданный в кавычках, и система будет ждать ввода точки или вызова контекстного меню. Если вызвать контекстное меню, то в нем появятся пункты, перечисленные в квадратных скобках (рис. 4.5).

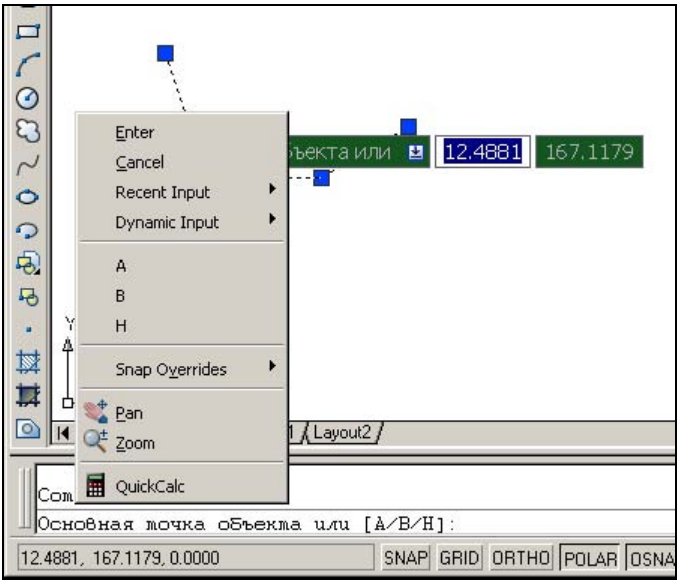


Рис. 4.5. Связь контекстного меню с опциями выполняемой команды

Проверку опции, выбранной пользователем, следует выполнять с использованием функции `initget`, которая не дает возможности ввода неразрешенных ключевых слов и их сокращений.

4.9. MNL-файлы

Одновременно с загрузкой файла шаблона меню или адаптации система AutoCAD пытается загрузить также MNL-файл с таким же именем, что и у файлов меню. Например, в версии 2005 или более ранней при загрузке файла acad.mnu (или файлов acad.mnc, acad.mns, acad.mnr) загружается файл acad.mnl, который является файлом на языке AutoLISP. В версии 2006 то же относится к загрузке файла acad.cui.

В листинге 4.1 приведен фрагмент файла acad.mnl версии 2006.

Листинг 4.1. Фрагмент файла acad.mnl версии 2006

```
;;;     AutoLISP routines used by the AutoCAD Standard Menu.
;;;
;;;     This file is loaded automatically following the menu ACAD.

(princ "\nAutoCAD menu utilities ")

;;;=== Icon Menu Functions ===
;;;  View -> Layout -> Tiled Viewports...
```

```
(defun ai_tiledvp_chk (new)
  (setq m:err *error* *error* *merrmsg*)

  (if (= (getvar "TILEMODE") 0)
    (progn
      (princ "\n** Command not allowed in a Layout **")
      (princ)
    )
    (progn
      (if new
        (menucmd "I=ACAD.IMAGE_VPORTI")
        (menucmd "I=IMAGE_VPORTI")
      )
      (menucmd "I=*")
    )
  )
  (setq *error* m:err m:err nil)
  (princ)
)

(defun ai_tiledvp (num ori / ai_tiles_g ai_tiles_cmde)
  (setq m:err *error* *error* *merrmsg*
        ai_tiles_cmde (getvar "CMDECHO")
        ai_tiles_g (getvar "GRIDMODE")
  )
  (ai_undo_push)
  (setvar "CMDECHO" 0)
  (setvar "GRIDMODE" 0)
  (cond ((= num 1)
         (command "_.VPORTS" "_SI")
         (setvar "GRIDMODE" ai_tiles_g)
        )
. . . . . .
```

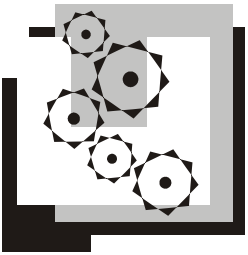
Из листинга видно, что это обычный файл, который может загружать необходимые описания функций или выполнять некоторые выражения на языке AutoLISP (просматривается аналогия с файлом acaddoc.lsp, который загружается при открытии нового рисунка).

Поэтому пользователь, создавая свой файл меню (например, mynew.mnu или mynew.cui), может написать файл mynew.mnl, в который включить необходимые LISP-программы или LISP-выражения. Такой файл следует поместить в ту же папку, что и файл меню, и они будут загружаться один за другим.

4.10. ActiveX

Более интересными средствами работы с меню являются функции ActiveX, взаимодействующие с объектами (элементами объектной модели) `MenuGroup`, `PopupMenu`, `PopupMenuItem`, `Toolbar`, `ToolbarItem`. Работа с такими объектами позволяет динамически вносить изменения в меню (добавлять, удалять, изменять отдельные меню или их пункты). Эти более мощные средства работы с меню и адаптациями рассматриваются в *главах 6—8*.

ГЛАВА 5



Справочные системы

Современный уровень разработки приложений требует включения в их состав достаточно подробной справочной системы. Такая система помогает пользователю в освоении программного комплекса. Чем лучше изложена справочная информация, тем легче разработчику осуществлять сопровождение своего продукта.

AutoCAD позволяет включать в приложения справочные файлы следующих типов:

- HELP-файлы Windows;
- HTML-файлы;
- справочные файлы предыдущих версий AutoCAD (устаревшие).

В данной главе рассматриваются вопросы работы с этими файлами, а также другие вопросы вывода справочной информации.

5.1. HELP-файлы Windows

Самые привычные и несколько устаревшие справочные файлы — это HELP-файлы (WinHelp). Такие файлы могут просматриваться пользователем как при работе в системе AutoCAD, так и вне системы. Файлы типа WinHelp могут помимо текста содержать графические иллюстрации, макросы, мультимедийные элементы, что позволяет сделать справочное руководство удобным и привлекательным. Для создания этих файлов требуются специальные пакеты (Microsoft Help Workshop, Microsoft Help Designer и т. п.).

5.1.1. Структура HELP-файлов

В состав HELP-файлов Windows входят файлы со следующими расширениями:

- rtf (текстовые RTF-файлы);
- hpj (файл проекта справочной системы);

- hlp (компилированный HLP-файл);
- cnt (файл разделов справки);
- gid (файл конфигурации справочной системы);
- fts (файл полного поиска, образуется при первом обращении к базе справочных терминов);
- ftg (файл полного поиска групп).

Основными файлами, которые передаются пользователю при установке программного продукта с HELP-файлами, являются файлы с расширениями hlp и cnt (в простых случаях поставляются только файлы с расширениями hlp). Например, в папке Help программного обеспечения системы AutoCAD 2006 находятся файлы с расширением hlp (drv_ccp.hlp, drv_dwf.hlp, drv_oce.hlp и т. д.) и файлы с расширением cnt (drv_ccp.cnt, drv_oce.cnt и т. д.).

Файлы с расширением hlp получаются путем компиляции проектов в специальных пакетах (Microsoft Help Designer и др.). Файлы с расширением cnt являются текстовыми. Они тоже могут быть сформированы с помощью специального пакета, но могут быть написаны или отредактированы и обычным текстовым редактором.

На рис. 5.1 приведено окно HELP-системы описания драйверов для плоттеров фирмы OCE.

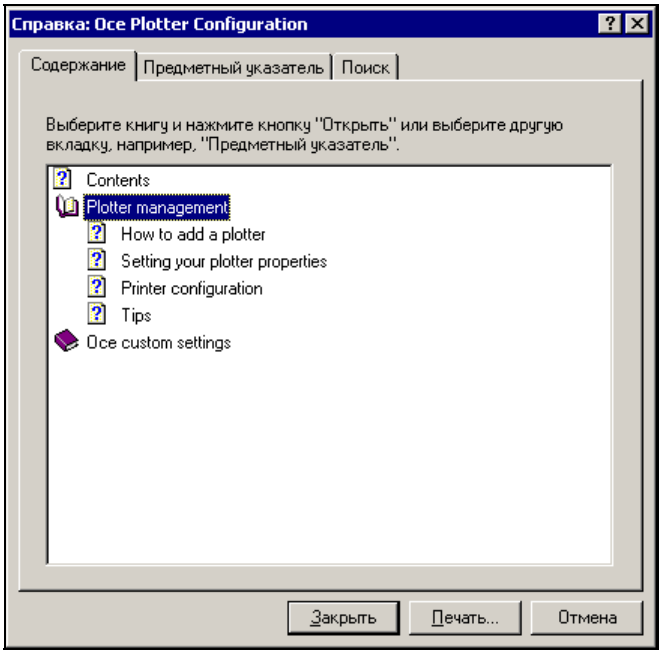


Рис. 5.1. Окно HELP-системы

Данное диалоговое окно имеет три вкладки. Вкладка **Содержание** (Contents) показывает все разделы верхнего уровня справочной системы, вкладка **Предметный указатель** (Index) является предметным указателем, а вкладка **Поиск** (Find) используется для более глубокого поиска по базе терминов.

Справочные разделы, приведенные на рис. 5.1, могут быть сформированы из разных справочных файлов, однако выглядеть как единая система. Управление итоговой формой справочного диалогового окна осуществляется через файлы с расширением cnt, которые содержат наименования разделов справочной информации и имена файлов с расширением hlp, где эти разделы физически размещены.

5.1.2. CNT-файлы

В листинге 5.1 приведен текст файла drv_oce.cnt версии 2006.

Листинг 5.1. Файл drv_oce.cnt

```
:Base DRV_OCE.HLP
:Title Oce Plotter Configuration
1 Contents=IDH_contents
1 Plotter management
2 How to add a plotter=IDH_adding_plotter
2 Setting your plotter properties=IDH_setting_plotter_properties
2 Printer configuration=IDH_printer_configuration
2 Tips=hints_oce
1 Oce custom settings
2 Introduction=IDH_intro_custom_settings
2 Remote control header=IDH_remote_control_header
2 Basic option group=IDH_basic
2 Transformation option group=IDH_transformation
2 Finishing option group=IDH_finishing_group
2 Layout option group=IDH_layout_group
2 Stamping option group=IDH_stamping_option_group
2 Plot delivery option=IDH_delivery_of_plots
2 Accounting option=IDH_accounting_option
```

В CNT-файле пустые строки и строки, начинающиеся с точки с запятой, являются комментариями. Внутренним операторам предшествует символ двоеточия. Основные типы внутренних операторов следующие:

- `:Base` — имя основного файла с расширением hlp (с этим файлом связываются другие аналогичные файлы);
- `:Title` — заголовок, выносимый в диалоговое окно;
- `:Index` — добавление элементов указателя;

- `:Link` — связывание с другим HELP-файлом;
- `:Include` — добавление к содержанию элементов из другого CNT-файла.

Если строка CNT-файла начинается с числа, то это число определяет уровень заголовка в содержании. Уровень 1 является самым верхним, уровень 2 — на один уровень ниже уровня 1 и т. д. Заголовки уровня 2 должны следовать сразу за заголовком уровня 1, которому они подчиняются. Аналогичное требование предъявляется к заголовкам уровня 3 относительно уровня 2 и т. д.

Если пользователь хочет добавить свои разделы, для этого ему предоставляется возможность написать собственные CNT-файлы и указать в основном CNT-файле подключение своих разделов, например:

```
:Include spravka1.cnt
:Include spravka2.cnt
```

Упоминаемые файлы spravka1.cnt и spravka2.cnt должны быть расположены в папках, указанных в настройке путей файлов поддержки.

5.1.3. Функция *help*

Основным средством вызова HELP-файла из AutoLISP является функция `help`:

```
(help [<файл> [<раздел> [<начало>]]])
```

Аргументы функции `help`: `<файл>` — имя справочного файла (с расширением или без него), `<раздел>` — наименование раздела справки, `<начало>` — параметр, управляющий выбором начальной страницы при загрузке справочного окна.

Если аргумент опущен или в качестве его значения `<файл>` указана пустая строка (`""`), то система AutoCAD вызывает справочный файл, указанный в переменной окружения ACADHELP, а затем — файл, указанный в настройке как файл справки по умолчанию. Если в аргументе `<файл>` опущено расширение, то AutoCAD сначала ищет файл с расширением chm, затем — с hlp.

В качестве аргумента `<раздел>` задается имя раздела (из CNT-файла), который нужно сразу же показать при вызове справки. Если аргумент `<раздел>` опущен, задан неправильно или задан равным пустой строке, то выводится первый раздел справочного файла.

Аргумент `<начало>` для HELP-файлов может принимать одно из трех значений:

- `"HELP_CONTENTS"` — показывает первый раздел справки в файле;
- `"HELP_HELPONHELP"` — вызывает справку Windows о пользовании справочной системой;
- `"HELP_PARTIALKEY"` — активизирует вкладку **Предметный указатель** (Index) с отметкой на параметре, заданном в функции `help` в качестве аргумента `<раздел>`; если такого раздела нет, то подбирается ближайший по наименованию.

Функция help возвращает значение аргумента <файл> при успешном выполнении операции и nil — при неуспешном. Если функция вызывалась без параметров, то возвращается "" или nil.

5.1.4. Вызов WIN-справки с помощью функции *startapp*

Функция startapp может быть полезной при вызове справки, для чего нужно запустить специальную справочную программу Windows с именем Winhlp32, а в качестве второго аргумента передать полное имя справочного файла:

```
(startapp "Winhlp32" "d:\\Temp\\new_conditions.hlp")
```

Если имя файла содержит пробелы (например, d:\Temp\new conditions.hlp), то в состав второго аргумента необходимо включить обрамляющие двойные кавычки:

```
(startapp "winhlp32" "\"d:\\Temp\\new conditions.hlp\"")
```

5.2. HTML-справки

Если необходимо оформить разделы справочной информации в виде Web-страниц, то следует использовать файлы в формате HTML (расширения файлов — chm, htm, html). Большинство систем разработки справочных файлов обеспечивает создание как HELP-, так и HTML-файлов.

5.2.1. Вызов HTML-справки с помощью функции *help*

Для вызова справки из файла с расширением chm можно пользоваться стандартной функцией help, например:

- (help "acad162.chm") или (help "acad162") — открывает окно справки системы AutoCAD 2006 (рис. 5.2).

5.2.2. Вызов HTML-справки с помощью функции *startapp*

Для вызова в среде AutoCAD справочного HTML-файла можно с помощью функции startapp открыть приложение, читающее эти файлы.

Пример:

- ```
  (setq webrdr "\"C:\\Program Files\\Internet Explorer\\IEXPLORE.EXE\"")
  (startapp webrdr "d:\\Temp\\our rules.htm")
  ```

В первой строке этого примера функция setq запоминает в переменной webrdr полное имя программы (Internet Explorer), причем нужно использовать дважды
  ```

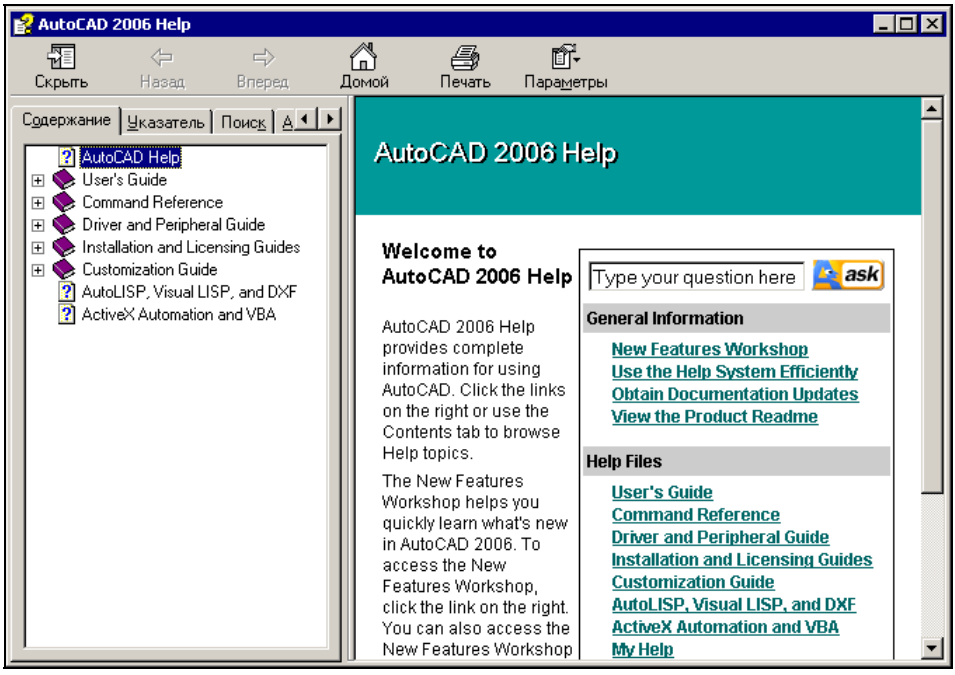


Рис. 5.2. Окно справочной системы AutoCAD 2006

двойные кавычки. Затем функция `startapp` вызывает приложение и с его помощью открывает HTML-файл.

5.3. Устаревшие справочные файлы AutoCAD

К устаревшим справочным файлам системы AutoCAD относятся файлы с расширением ahp, которые использовались в предыдущих версиях (до 2000).

5.3.1. Функция *acad_helpdlg*

Для временной поддержки старых справочных файлов в перечне функций языка AutoLISP сохранена следующая функция:

```
(acad_helpdlg <файл> <раздел>)
```

Оба аргумента являются текстовыми строками (задают имя справочного файла с расширением ahp и имя раздела справки). По своему действию функция аналогична функции `help`.

5.4. Контекстно-зависимая справка к командам пользователя

При написании команд пользователя удобно по клавише <F1> получать контекстно-зависимую справку, отражающую во время работы команды описание ее действия и опций. Стандартные команды системы AutoCAD такому условию отвечают. Например, если после начала работы команды LINE (ОТРЕЗОК) вы в ответ на запрос ввода первой или второй точек нажмете клавишу <F1>, то откроется справочный раздел, посвященный построению отрезков с помощью этой команды.

5.4.1. Функция *setfunhelp*

Функция `setfunhelp` дает возможность сгенерировать контекстно-зависимую справку для команд, созданных пользователем. Синтаксис функции:

```
(setfunhelp <функция-команда> [<файл> [<раздел> [<начало>]]])
```

Аргументы: `<функция-команда>` — имя функции, начинающееся с префикса `C:` или `c:` и определяющее новую команду AutoCAD; `<файл>` — имя справочного файла; `<раздел>` — наименование раздела справочного файла; `<начало>` — начальная страница справки. Все аргументы — текстовые строки.

В качестве имени функции, вводимой вместо первого аргумента, должно быть имя с префиксом `C:` или `c:` (оба начинаются с латинской буквы C). Если, например, функция называется `c:crown`, то она определяет новую команду CROWN.

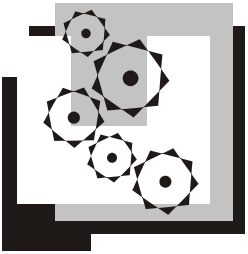
Второй, третий и четвертый аргументы аналогичны аргументам функции `help` *(см. разд. 5.1.3)*. Если в аргументе `<файл>` опущено расширение, то AutoCAD сначала ищет файл с расширением chm, затем — с расширением hlp.

При успешном завершении функция `setfunhelp` возвращает значение первого аргумента, при неудачном — `nil`.

5.4.2. Особенности применения функции *setfunhelp*

Как было сказано в *главе 1*, пользовательские функции создаются, например, при помощи функции `defun`. Описание любой пользовательской функции должно обязательно предшествовать в программном файле указанию имени этой функции в качестве первого аргумента в функции `setfunhelp`. В противном случае результат работы программы не предсказуем.

ГЛАВА 6



Технология ActiveX

В данной главе рассматриваются функции LISP, входящие в расширение Visual LISP и реализующие в системе AutoCAD технологию ActiveX, разработанную фирмой Microsoft.

Технология ActiveX (точнее, ActiveX Automation, а ранее она называлась OLE Automation) реализует идеологию объектно-ориентированного программирования и позволяет оперировать не байтами, числами, а объектами прикладной среды и работать с ними на языке, подобном языку обычных манипуляций. Такая идеология внедрена в самые распространенные прикладные пакеты — например, Microsoft Word, Microsoft Excel и др. Она значительно облегчает обмен данными между файлами, созданными разными системами.

6.1. Расширение языка AutoLISP

В технологии ActiveX объектная модель работающего приложения представляется как совокупность объектов, свойств, методов и событий. Для каждого из элементов этой модели имеется своя реализация в виде данных новых типов (структур) и функций, обеспечивающих взаимодействие с пользователем.

6.1.1. Объекты и семейства

Объекты (objects) ActiveX в системе AutoCAD рассматриваются как иерархия, содержащая не только примитивы, но и прочие элементы (таблицы, словари и т. д.). Однотипные объекты объединяются в *семейства* (collections). Корневым элементом этой иерархии является объект `Application`, остальные находятся на более низких уровнях.

Корневой объект `Application` связан с семействами:

- `Documents` (Документы);
- `MenuBar` (Строка меню);

- `MenuGroups` (Группы меню или адаптаций);
- `Preferences` (Настройки).

Семейство `Documents` включает объекты типа `Document` — открытые документы (рисунки). В семейство `MenuGroups` входят объекты типа `MenuGroup` (Группы меню или адаптаций), а в семейство `MenuBar` — объекты типа `PopupMenu` (Падающие и контекстные меню). В семейство `Preferences` в качестве объектов входят различные настройки (файловые, чертежные и т. п.).

Эта иерархия раскрывается дальше. В `MenuGroups` входят объекты `MenuGroup`, каждый из которых содержит семейства `PopupMenus` и `Toolbars`. Далее следуют цепочки:

- `PopupMenus` — `PopupMenu` — `PopupMenuItem`;
- `Toolbars` — `Toolbar` — `ToolbarItem`.

Полное иерархическое дерево системы AutoCAD приведено в документации разработчика по VBA[1] и в книге "AutoCAD: разработка приложений, настройка и адаптация"[2]. В *главах 7 и 8* настоящей книги приведены описания всех методов и свойств элементов объектной модели и даны примеры их использования.

Остановимся подробнее на объекте `Document`. Значимость этого объекта связана с тем, что именно в пространство текущего рисунка следует загрузить выполняемое LISP-приложение.

Объект `Document` содержит семейства:

- `Blocks` (Блоки);
- `ModelSpace` (Пространство модели);
- `PaperSpace` (Пространство листа);
- `Dictionaries` (Словари);
- `DimStyles` (Размерные стили);
- `Groups` (Группы);
- `Layers` (Слои);
- `Layouts` (Листы);
- `Linetypes` (Типы линий)

  и т. д.

В первые три семейства входят графические объекты, которыми оперирует система AutoCAD (имена объектов аналогичны типам примитивов). Кроме того, сю-

---

[1] ActiveX and VBA Reference.

[2] Полещук Н. Н. AutoCAD: разработка приложений, настройка и адаптация. — СПб.: БХВ-Петербург, 2006.— 992 с. (+CD-ROM).

да же входят семейство Hyperlinks и такие неграфические объекты, как Layer, Linetype и т. п. В последних версиях системы появились дополнительные объекты, связанные с базами данных и подшивками листов. В табл. 6.1 приведен перечень объектов и семейств по состоянию версии 2006. Объекты, существующие только в AutoCAD 2006, отмечены цифрами в скобках.

Таблица 6.1. *Объекты и семейства*

Имя объекта	**Описание**
3DFace	3М-грань
3DPolyline	3М-полилиния
3DSolid	Тело
AcadState	Состояние системы AutoCAD как приложения
AcCmColor	Точный цвет (из полной палитры)
AcSmAcDbBlockRecordReference	Ссылка на запись блока
AcSmAcDbDatabase	База, связанная с подшивкой
AcSmAcDbLayoutReference	Ссылка на вкладку рисунка
AcSmAcDbObjectReference	Ссылка на компонент AcDbObject
AcSmAcDbViewReference	Ссылка на вид
AcSmCalloutBlockReferences	Ссылки на блоки-идентификаторы (семейство)
AcSmCalloutBlocks	Блоки-идентификаторы (семейство)
AcSmCustomPropertyBag	Контейнер пользовательских свойств
AcSmCustomPropertyValue	Значение пользовательского свойства
AcSmDatabase	Компонент базы
AcSmDSTFiler	Обработчик DST-файлов
AcSmEvent	Структура (перечисление), содержащая номера событий
AcSmFileReference	Ссылка на файл
AcSmNamedAcDbObjectReference	Ссылка на именованный объект
AcSmObjectReference	Ссылка на объект
AcSmPersistProxy	Постоянный прокси-объект
AcSmPublishOptions	Опции публикации подшивки
AcSmResources	Ссылки на файлы ресурсов (семейство)
AcSmSheet	Лист подшивки
AcSmSheetSelSet	Набор листов подшивки
AcSmSheetSelSets	Наборы листов подшивки (семейство)

Таблица 6.1 *(продолжение)*

Имя объекта	Описание
AcSmSheetSet	Подшивка
AcSmSheetSetMgr	Диспетчер подшивок
AcSmSheetView	Вид на листе
AcSmSheetViews	Виды на листе (семейство)
AcSmSubset	Группа листов
AcSmViewCategories	Категории видов (семейство)
AcSmViewCategory	Категория видов
Application	Приложение
Arc	Дуга
Attribute	Описание атрибута
AttributeReference	Атрибут
Block	Описание блока
BlockRef	Блок (вхождение блока)
Blocks	Блоки (семейство)
Body	Лекальная поверхность (напрямую в системе AutoCAD не создается)
Circle	Круг
Database	Графическая база рисунка
DbConnect	Соединение с внешней БД
DatabasePreferences	Настройки рисунка
Dictionaries	Словари (семейство)
Dictionary	Словарь
Dim3PointAngular	Угловой размер по трем точкам
DimAligned	Выровненный размер
DimAngular	Угловой размер
DimArcLength (2006)	Размер длины дуги
DimDiametric	Диаметральный размер
DimOrdinate	Ординатный размер
DimRadial	Радиальный размер
DimRadialLarge (2006)	Радиальный размер со сломом размерной линии
DimRotated	Повернутый размер

Таблица 6.1 *(продолжение)*

Имя объекта	Описание
DimStyle	Размерный стиль
DimStyles	Размерные стили (семейство)
Document	Документ (рисунок)
Documents	Документы (семейство)
DynamicBlockReferenceProperty (2006)	Динамическое свойство вхождения блока
Ellipse	Эллипс
ExternalReference	Внешняя ссылка
FileDependencies	Внешние зависимости (семейство)
FileDependency	Элемент из списка файловых зависимостей
Group	Группа
Groups	Группы (семейство)
Hatch	Штриховка
Hyperlink	Гиперссылка
Hyperlinks	Гиперссылки (семейство)
IDPair	Специальный объект метода CopyObjects
KeyDescription	Ключевой столбец
KeyDescriptions	Ключевые столбцы (семейство)
KeyValue	Ключевое значение
KeyValues	Ключевые значения (семейство)
Label	Ярлык
LabelTemplate	Шаблон ярлыка
LabelTemplates	Шаблоны ярлыка (семейство)
Layer	Слой
Layers	Слои (семейство)
LayerStateManager	Диспетчер свойств слоев
Layout	Вкладка
Layouts	Вкладки (семейство)
Leader	Выноска
LightweightPolyline	Компактная двумерная полилиния
Line	Отрезок

Таблица 6.1 *(продолжение)*

Имя объекта	Описание
Linetype	Тип линий
Linetypes	Типы линий (семейство)
Link	Связь
Links	Связи (семейство)
LinkTemplate	Шаблон связи
LinkTemplates	Шаблоны связи (семейство)
MenuBar	Строка меню (семейство)
MenuGroup	Группа меню (до 2005) или адаптации (2006)
MenuGroups	Группы меню (до 2005) или адаптации (2006) (семейство)
MinsertBlock	Массив блоков
Mline	Мультилиния
ModelSpace	Пространство модели (семейство)
Mtext	Мультитекст
Ole	Контейнер OLE-объекта
PaperSpace	Пространство листа (семейство)
Plot	Набор свойств печати листа
PlotConfiguration	Настройка печати
PlotConfigurations	Настройки печати (семейство)
Point	Точка
PolyfaceMesh	Многогранная сеть (сеть из полиграней)
PolygonMesh	Полигональная сеть
Polyline	Подробная двумерная полилиния
PopupMenu	Падающее или контекстное меню
PopupMenuItem	Пункт падающего или контекстного меню
PopupMenus	Падающие и контекстные меню (семейство)
Preferences	Настройки системы AutoCAD
PreferencesDisplay	Экранные настройки
PreferencesDrafting	Настройки рисования
PreferencesFiles	Файловые настройки
PreferencesOpenSave	Настройки открытия и сохранения

Таблица 6.1 *(продолжение)*

Имя объекта	Описание
PreferencesOutput	Настройки вывода
PreferencesProfiles	Настройки профилей
PreferencesSelection	Настройки выбора
PreferencesSystem	Системные настройки
PreferencesUser	Пользовательские настройки
PropertyFlags	Флаги пользовательских свойств
ProxyEntity	Объект-заместитель
PViewport	Видовой экран пространства листа
Queries	Запросы (семейство)
Query	Запрос
Raster	Растровое изображение
Ray	Луч
Region	Область
RegisteredApplication	Зарегистрированное приложение
RegisteredApplications	Зарегистрированные приложения (семейство)
SecurityParams	Настройки безопасности
SelectionSet	Набор выбора
SelectionSets	Наборы (семейство)
Shape	Форма
Solid	Фигура
SortentsTable	Таблица управления порядком следования
Spline	Сплайн
SummaryInfo	Свойства рисунка (автор, заголовок и т. п.)
Table	Таблица
TableStyle	Стиль таблиц
TableStyles	Стили таблиц (семейство)
Text	Текст (однострочный)
TextStyle	Текстовый стиль
TextStyles	Текстовые стили (семейство)

Таблица 6.1 *(окончание)*

Имя объекта	Описание
Tolerance	Допуск
Toolbar	Панель инструментов
ToolbarItem	Элемент панели инструментов
Toolbars	Панели инструментов (семейство)
Trace	Полоса
UCS	ПСК
UCSs	Семейство ПСК
Utility	Утилита
VBE	Среда VBA
View	Вид
Viewport	Видовой экран
Viewports	Видовые экраны (семейство)
Views	Виды (семейство)
XLine	Прямая
XRecord	X-запись

Работа с каждым из таких объектов в технологии ActiveX ведется на уровне указателей (адресов). Для поддержки объектов разработаны функции, входящие в Visual LISP — расширение языка AutoLISP. Все эти функции имеют префиксы `vla-`, `vlax-` или `vlr-` (минус тоже входит в префиксы).

Однако, в отличие от обычных функций, функции ActiveX не всегда доступны сразу же после начала сеанса системы AutoCAD. Для их загрузки следует выполнить функцию `vl-load-com`, рассмотренную в *разд. 6.1.2*.

В настоящее время объекты подшивок листов (с префиксами `AcSm`) не доступны для работы из Visual LISP.

6.1.2. Функция *vl-load-com*

Данная функция загружает большое количество функций, входящих в расширение AutoLISP, позволяющее работать с объектами, свойствами, методами и событиями ActiveX.

```
(vl-load-com)
```

Возвращаемого значения у функции нет.

6.1.3. Свойства

Каждый объект в иерархии, используемой технологией ActiveX, имеет свои *свойства*, которые можно получить с помощью соответствующих функций расширения AutoLISP.

Для получения доступа к такому объекту нужно сначала пройти по иерархии модели объектов AutoCAD и прочитать указатели всех объектов более высоких уровней (объекта `Application`, документа, пространства модели). Четыре функции, позволяющие прочитать эти указатели, чтобы сохранить их в выделенных пользователем переменных, рассматриваются в *разд. 6.1.4—6.1.7*.

Имена всех функций могут вводиться в нижнем или верхнем регистрах, поскольку AutoLISP безразличен к регистру (в этом его отличие от ObjectARX).

Свойства объектов иерархической модели индивидуальны. Помимо традиционных характеристик (color, layer), к свойствам объектов относятся и такие, как `radius`, `area`, `length`, `perimeter` и т. д. Для их чтения имена соответствующих функций будут `vla-get-color`, `vla-get-radius`, `vla-get-area` и т. д.

К операциям получения свойств относятся и операции получения дочерних объектов. Например, для объекта `Application` функция `vla-get-Documents` возвращает указатель на семейство `Documents` (открытые документы), а функция `vla-get-ActiveDocument` возвращает указатель на активный документ. Получить список всех свойств объекта или семейства можно с помощью функции `vlax-dump-object` *(см. разд. 6.1.25)*.

Полный перечень всех свойств с подробностями приведен в *главе 8*. Имена функций свойств могут быть получены из имен свойств добавлением префикса `vla-get-`. Другие функции, имеющие префикс `vla-put-` (например, `vla-put-Color`, `vla-put-Layer`), позволяют изменять свойства объектов.

Работа с функциями с префиксами `vla-get-` и `vla-put-` будет коротко рассмотрена в *разд. 6.1.13* и подробно — в *главе 8*.

6.1.4. Функция *vlax-get-acad-object*

Читает указатель корневого объекта иерархии AutoCAD (имя объекта — `Application`).

```
(vlax-get-acad-object)
```

Возвращаемое значение — VLA-объект (указатель объекта `Application`).

Пример возвращаемого значения: `#<VLA-OBJECT IAcadApplication 00adc088>`.

О VLA-объектах *см. разд. 6.1.10*.

6.1.5. Функция *vla-get-ActiveDocument*

Читает указатель открытого документа (активного рисунка).

```
(vla-get-activedocument <корень>)
```

Аргумент `<корень>` — указатель корневого объекта, возвращенный функцией `vlax-get-acad-object`.

Возвращаемое значение — VLA-объект (указатель открытого документа).

Пример возвращаемого значения: `#<VLA-OBJECT IAcadDocument 00f6d534>`.

6.1.6. Функция *vla-get-ModelSpace*

Читает указатель объекта `ModelSpace` (пространства модели рисунка).

```
(vla-get-modelspace <документ>)
```

Аргумент `<документ>` — указатель документа, возвращенный функцией `vlax-get-activedocument`.

Возвращаемое значение — VLA-объект (указатель пространства модели).

Пример возвращаемого значения: `#<VLA-OBJECT IAcadModelSpace 01923224>`.

6.1.7. Функция *vla-get-PaperSpace*

Читает указатель объекта `PaperSpace` (пространства листа документа).

```
(vla-get-paperspace <документ>)
```

Аргумент `<документ>` — указатель документа, возвращенный функцией `vlax-get-activedocument`.

Возвращаемое значение — VLA-объект (указатель пространства листа).

Пример возвращаемого значения: `#<VLA-OBJECT IAcadPaperSpace 01920344>`.

6.1.8. Подготовка к работе с ActiveX

VLA-объекты, полученные функциями, рассмотренными в *разд. 6.1.4—6.1.7*, очень важны. Их можно сохранить в глобальных переменных, чтобы не обращаться к этим функциям повторно. В листинге 6.1 приведен текст функции `begin_activex`, к которой желательно обращаться перед работой приложений, использующих технологию ActiveX.

Листинг 6.1. Функция `begin_activex`

```
; Глава 06\Book01\begin_activex.lsp
(defun begin_activex ( / )
  (vl-load-com)
  (setq acad_application (vlax-get-acad-object))
  (setq active_document (vla-get-ActiveDocument acad_application))
  (setq model_space (vla-get-ModelSpace active_document))
  (setq paper_space (vla-get-PaperSpace active_document))
); defun begin_activex
```

Предпоследняя строка в вашем варианте функции `begin_activex` может быть опущена, если вам не потребуются объекты из пространства листа.

6.1.9. Методы

По идеологии ActiveX каждому объекту соответствует перечень *методов* (операций), которые могут применяться к объекту (не всегда это надо понимать в прямом смысле). Например, активный документ можно закрыть.

Для объектов каждого типа из входящих в семейство `ModelSpace` обязательно существует метод, позволяющий создавать новые объекты. Имя соответствующей функции LISP, создающей объекты этого типа, образуется по такому правилу: к префиксу `vla-Add` добавляется наименование типа объекта из перечисленных в *разд. 6.1.1*, например: `vla-Add3Dface`, `vla-Add3Dpoly`, `vla-Add3Dsolid`, `vla-AddArc`, `vla-AddAttribute`, `vla-AddAttributeRef`, `vla-AddBlockRef`, `vla-AddCircle`, `vla-AddDim3PointAngular` и т. д.

Все символы наименований функций могут быть в нижнем или в верхнем регистре. Количество передаваемых функциям аргументов индивидуально для каждой из них. Оно определяется описанием метода (имена методов — `Add3DFace`, `Add3DPoly`, `AddCircle` и т. д.).

Существует много других методов (и реализующих их в языке LISP функций), применяемых к объектам. Каждой команде системы AutoCAD сопоставляются методы с теми же именами. Например, командам COPY (КОПИРОВАТЬ) и MIRROR (ЗЕРКАЛО) сопоставлены методы `Copy` и `Mirror`. Этим методам соответствуют функции `vla-Copy` и `vla-Mirror` (получаются добавлением префикса `vla-`).

Могут быть и отличия. Так, команде ARRAY (МАССИВ) сопоставляются два метода (`ArrayRectangular` и `ArrayPolar`), поэтому указанной команде соответствуют две функции: `vla-ArrayRectangular` и `vla-ArrayPolar`.

Полный перечень методов с подробными примерами *см. в главе 7*. Получить список всех методов объекта или семейства можно с помощью функции `vlax-dump-object` *(см. разд. 6.1.25)*.

6.1.10. VLA-объекты

Объекты иерархии ActiveX называются также *VLA-объектами*. Функции `vlax-get-acad-object`, `vla-get-ActiveDocument`, `vla-get-ModelSpace`, `vla-get-PaperSpace`, а также все функции с префиксами `vla-Add` возвращают VLA-объекты. Фактически VLA-объекты — это указатели (адреса) в формируемой в оперативной памяти базе данных, включающей сам AutoCAD, открытые документы, объекты рисунков и сопутствующие данные. Память, выделяемую динамически под VLA-объекты, желательно освобождать *(см. разд. 6.3)*.

Возможно преобразование (сопоставление) обычных объектов в VLA-объекты и обратно. Такие преобразования осуществляют функции `vlax-ename->vla-object` и `vlax-vla-object->ename`, рассматриваемые в *разд. 6.1.11* и *6.1.12*.

6.1.11. Функция *vlax-ename->vla-object*

Преобразует обычный примитив в VLA-объект.

```
(vlax-ename->vla-object <примитив>)
```

Аргумент функции — примитив в том виде, в каком он возвращается функцией `entlast`.

Возвращаемое значение — VLA-объект.

В качестве примера построим, например, в рисунке отрезок и преобразуем его в VLA-объект:

- `(setq vlo1 (vlax-ename->vla-object (entlast)))` возвратит `#<VLA-OBJECT IAcadLine 01925274>` (или что-то похожее)

6.1.12. Функция *vlax-vla-object->ename*

Преобразует VLA-объект в обычный примитив.

```
(vlax-vla-object->ename <объект>)
```

Аргумент `<объект>` — VLA-объект (указатель) в том виде, в каком он возвращается функцией `vlax-ename->vla-object`.

Возвращаемое значение — примитив.

Пример (обратный к примеру из *разд. 6.1.11*):

- `(setq e1 (vlax-vla-object->ename vlo1))` возвратит `<Entity name: 4002ed88>` (или нечто аналогичное)

6.1.13. Работа со свойствами

Для VLA-объектов применимы функции чтения свойств (с префиксом `vla-get-`) и редактирования свойств (с префиксом `vla-put-`). Например, для объекта `vlo1`, созданного в *разд. 6.1.11* с помощью функции `vlax-ename->vla-object`, можно прочитать его основные свойства:

- `(vla-get-Color vlo1)` вернет `1`;
- `(vla-get-Layer vlo1)` вернет `"0"`;
- `(vla-get-Linetype vlo1)` вернет `"ByLayer"`.

Зная, что свойства `StartPoint` и `EndPoint` отрезка хранят координаты его первой и второй точек, можно получить их с помощью функций ActiveX:

- `(setq va1 (vla-get-StartPoint vlo1))` вернет `#<variant 8197 ...>`
- `(setq va2 (vla-get-EndPoint vlo1))` вернет `#<variant 8197 ...>`

В обоих примерах в качестве возвращаемого значения получены варианты — данные нового типа, рассматриваемые в следующем разделе.

Функции, имеющие вместо символов `get` символы `put`, выполняют редактирование VLA-объектов, например:

- `(vla-put-Color vlo1 3)` изменяет цвет объекта `vlo1` на зеленый

6.1.14. Варианты

При создании VLA-объектов с помощью функций с префиксом `vla-Add` (например, `vla-AddCircle`) в качестве аргументов могут фигурировать данные ранее не известных в языке AutoLISP типов.

Метод `AddCircle` используется с двумя аргументами *(см. главу 7)*: центром и радиусом, причем центр должен быть так называемым *вариантом*, а радиус — вещественным числом с двойной точностью. Такие же аргументы должны передаваться и соответствующей этому методу функции `vla-AddCircle`. См. также уточнение в *разд. 6.1.24*.

Вариант — это структура, которая может хранить объекты разных типов. Создание варианта выполняется функцией `vlax-make-variant` *(см. разд. 6.1.15)*.

Три функции, работающие с вариантами (`vlax-variant-type`, `vlax-variant-value`, `vlax-variant-change-type`), рассматриваются в *разд. 6.1.16—6.1.18*.

Для создания вариантов с координатами трехмерных точек используется функция `vlax-3D-point` *(см. разд. 6.1.19)*. Такие варианты могут передаваться функциям, требующим для создания объектов точки (например, точка центра для окружности).

6.1.15. Функция *vlax-make-variant*

Создает вариант.

```
(vlax-make-variant [<значение> [<тип>]])
```

Аргументы: `<значение>` — значение, присваиваемое варианту; `<тип>` — целая константа, указывающая тип варианта. Если второй аргумент опущен, то `<тип>` система пытается определить по первому аргументу. Если оба аргумента опущены, то создается вариант типа `vlax-vbEmpty` (см. табл. 6.2).

Возвращаемое значение — вариант (указатель).

Зарезервированные константы, которые могут использоваться в качестве аргумента `<тип>`, приведены в табл. 6.2.

Таблица 6.2. *Константы, используемые в качестве типов вариантов*

Константа	Значение	Описание
vlax-vbEmpty	0	Неинициализированный (значение по умолчанию)
vlax-vbNull	1	Пустой (без данных)
vlax-vbInteger	2	Короткое целое число
vlax-vbLong	3	Длинное целое число
vlax-vbSingle	4	Вещественное число
vlax-vbDouble	5	Вещественное число с двойной точностью
vlax-vbString	8	Строка
vlax-vbObject	9	Объект
vlax-vbBoolean	11	Булево значение
vlax-vbArray	8192	Массив

Как и имена функций, имена констант не чувствительны к регистру. Поскольку в других реализациях числовые значения, присвоенные зарезервированным константам, могут меняться, то к ним следует обращаться по имени, а не по значению.

Примеры использования функции `vlax-make-variant`:

- `(vlax-make-variant nil)` возвращает `#<variant 0>`
- `(setq vr (vlax-make-variant 17.25 vlax-vbDouble))` возвращает `#<variant 5 17,25>`
- `(setq vs (vlax-make-variant "tools"))` возвращает `#<variant 8 tools>`

Если возникает необходимость создать вариант типа `vlax-vbArray`, то в качестве первого аргумента нужно передать данные нового типа — массива, который в ActiveX называется *безопасный массив* (safearray). О безопасных массивах *см. разд. 6.1.20*.

Если `sf` — безопасный массив, то следующее выражение создает вариант типа `vlax-vbArray`:

- `(setq vas (vlax-make-variant sf))` возвращает `#<variant 8197 ...>`

6.1.16. Функция *vlax-variant-type*

Создает вариант.

```
(vlax-variant-type <вариант>)
```

Аргумент `<вариант>` — вариант.

Возвращаемое значение — целое число, соответствующее значениям из второй колонки табл. 6.2. Если вариант имеет тип `vlax-vbArray`, то возвращается число вида 8192 + N, где N — число, обозначающее тип элементов массива.

Примеры (продолжение примеров из *разд. 6.1.15*):

- `(vlax-variant-type vr)` возвращает `5`
- `(vlax-variant-type vs)` возвращает `8`
- `(vlax-variant-type vas)` возвращает `8197`

6.1.17. Функция *vlax-variant-value*

Вычисляет значение варианта.

```
(vlax-variant-value <вариант>)
```

Аргумент `<вариант>` — вариант.

Возвращаемое значение — значение, присвоенное варианту.

Примеры (продолжение примеров из *разд. 6.1.15*):

- `(vlax-variant-value vr)` возвращает `17.25`
- `(vlax-variant-value vs)` возвращает `"tools"`
- `(vlax-variant-value vas)` возвращает `#<safearray...>`

6.1.18. Функция *vlax-variant-change-type*

Изменяет тип варианта, конвертируя значение.

```
(vlax-variant-change-type <вариант> <тип>)
```

Аргументы: `<вариант>` — изменяемый вариант, `<тип>` — целочисленная константа из первого столбца табл. 6.2.

Возвращаемое значение — вариант нового типа. Если конвертация невозможна, то возвращается `nil`.

- `(vlax-variant-change-type vr vlax-vbInteger)` возвращает `#<variant 2 17>` (вещественное число `17.25` округлено до целого `17`)
- `(vlax-variant-change-type vr vlax-vbString)` возвращает `#<variant 8 17,25>` (вещественное число `17.25` преобразовано в строку `"17,25"`)

6.1.19. Функция *vlax-3D-point*

Создает вариант из трех вещественных чисел двойной точности (может использоваться в качестве трехмерной точки). Обращение к функции может быть в двух видах:

```
(vlax-3D-point <список>) или (vlax-3D-point <x> <y> [<z>])
```

Аргументы: <список> — список из двух или трех вещественных чисел; <x>, <y>, <z> — вещественные числа.

Возвращаемое значение — вариант из трех вещественных чисел двойной точности (если координата *Z* не была задана, то она считается равной 0.0).

Примеры:

- `(setq vpt1 (vlax-3d-point 4 0 13.2))` возвращает `#<variant 8197 ...>` (вариант из трех вещественных чисел двойной точности)
- `(setq vpt1 (vlax-3d-point '(-12.4 6.0)))` возвращает `#<variant 8197 ...>` (тоже вариант из трех чисел с нулевой третьей координатой)

6.1.20. Безопасные массивы

Функции ActiveX имеют возможность работать с массивами (это тип данных, который в обычных функциях AutoLISP недоступен). Такие массивы называются *безопасными* (safearrays), поскольку система постоянно контролирует тип и количество сохраняемых в них числовых значений.

Для работы с безопасными массивами используются функции `vlax-make-safearray`, `vlax-safearray-put-element`, `vlax-safearray-fill`, рассматриваемые в *разд. 6.1.21—6.1.23*. Эти функции позволяют создавать, заполнять и изменять безопасные массивы.

6.1.21. Функция *vlax-make-safearray*

Создает безопасный массив размерности от 1 до 16 и инициализирует его.

```
(vlax-make-safearray <тип> '(<от1> . <до1>) ['(<от2> . <до2>) ...
['(<от16> . <до16>)]...])
```

Первый аргумент <тип> — это целочисленная константа. Все последующие аргументы имеют форму точечной пары, количество которых может быть от одного до шестнадцати — по величине размерности создаваемого массива. Все элементы точечных пар — целые числа (первый элемент — начальное значение индекса размерности массива, второй элемент — конечное значение индекса). Значения индексов могут быть как положительными, так и отрицательными, но значение конечного индекса не может быть меньше значения начального индекса.

В первом аргументе могут использоваться константы `vlax-vbInteger`, `vlax-vbLong`, `vlax-vbSingle`, `vlax-vbDouble`, `vlax-vbString`, `vlax-vbObject`, `vlax-vbBoolean` из первого столбца табл. 6.2, а также константа `vlax-vbVariant`, имеющая значение 12 и применяемая в качестве типа для вариантов.

Возвращаемое значение — безопасный массив (точнее, его указатель).

Значения, которыми инициализируются безопасные массивы, приведены в табл. 6.3.

Таблица 6.3. *Начальные значения элементов безопасных массивов*

Тип элемента	Начальное значение
vlax-vbInteger	0
vlax-vbLong	0
vlax-vbSingle	0.0
vlax-vbDouble	0.0
vlax-vbString	"" (пустая строка)
vlax-vbObject	nil
vlax-vbBoolean	:vlax-false (булева константа, используемая как "ложь")
vlax-vbVariant	Неинициализированный вариант (vlax-vbEmpty)

Пример:

- (setq sf (vlax-make-safearray vlax-vbBoolean '(-2 . 2))) создает одномерный массив из пяти булевых элементов и возвращает #<safearray...>

Элементы созданного в примере массива будут иметь такую нумерацию: –2, –1, 0, 1, 2.

6.1.22. Функция *vlax-safearray-put-element*

Присваивает значение элементу безопасного массива.

(vlax-safearray-put-element <бм> <индексы> <значение>)

Аргументы: <бм> — безопасный массив; <индексы> — целые числа (их количество равно размерности массива), указывающие номер элемента; <значение> — заносимое значение.

Возвращаемое значение — значение последнего аргумента.

Примеры (заполнение массива, созданного в *разд. 6.1.21*, булевыми константами):

- (vlax-safearray-put-element sf -2 :vlax-false) возвращает :vlax-false
- (vlax-safearray-put-element sf -1 :vlax-false) возвращает :vlax-false
- (vlax-safearray-put-element sf 0 :vlax-true) возвращает :vlax-true
- (vlax-safearray-put-element sf 1 :vlax-false) возвращает :vlax-false
- (vlax-safearray-put-element sf 2 :vlax-true) возвращает :vlax-true

6.1.23. Функция *vlax-safearray-fill*

Присваивает значения всем элементам безопасного массива.

(vlax-safearray-fill <бм> <список>)

Аргументы: <бм> — безопасный массив; <список> — список значений, присваиваемых элементам массива. Количество элементов и размерность аргумента <список> должны соответствовать размерам массива <бм>.

Возвращаемое значение — значение аргумента <бм>.

Пример (заполнение массива, созданного в *разд. 6.1.21*):

- `(vlax-safearray-fill sf '(:vlax-false :vlax-false :vlax-true :vlax-false :vlax-true))` возвращает `#<safearray...>`

6.1.24. Пример создания VLA-объекта

Попробуем в пространстве модели создать VLA-объект не преобразованием его из обычного примитива, а с помощью функции, соответствующей одному из методов с префиксом `Add` (префикс соответствующей функции будет `vla-Add`).

В *разд. 6.1.14* было сделано замечание, что с помощью функции `vla-AddCircle` можно создать окружность, передавая в качестве аргументов вариант с координатами трехмерной точки и вещественное число двойной точности (они должны стать центром и радиусом будущего круга). Сделаем уточнение: у функции есть еще один аргумент и он является первым — это VLA-объект, указывающий на пространство модели. Такой объект должен быть получен уже на стадии подготовки к работе с технологией ActiveX (см. объект `model_space` в листинге 6.1 *разд. 6.1.8*).

В качестве данных для построения окружности возьмем координаты центра (124.4, 25.9, 40.7) и радиус 100 мм (листинг 6.2). Используем функцию `begin_activex` из листинга 6.1.

Листинг 6.2. Построение окружности методом ActiveX

```
; Глава 06\Book01\sample.lsp
(begin_activex)
(setq center (vlax-3D-point '(124.4 25.9 40.7)))
(setq ax_circle (vla-AddCircle model_space center 100.0))
```

Второе выражение создает вариант с тремя координатами центра будущей окружности. Последнее выражение возвращает VLA-объект (`#<VLA-OBJECT IAcadCircle 0192cc04>`). Чтобы убедиться в его правильности, попробуем воспользоваться функцией `vlax-dump-object` (см. следующий раздел).

6.1.25. Функция *vlax-dump-object*

Выводит дамп объекта — полную распечатку свойств и, по запросу, перечня применимых к объекту методов.

```
(vlax-dump-object <объект> [<признак>])
```

Аргументы: <объект> — VLA-объект, <признак> — если задается и отличен от nil, то является указанием добавить в распечатку перечень методов, работающих с данным VLA-объектом.

Возвращаемое значение — T, если указано имя действительного VLA-объекта. В противном случае ничего не возвращается и может быть отображено сообщение об ошибке.

Функция выводит подробную распечатку свойств и методов объекта, которую можно назвать дампом.

Пример для объекта, созданного *разд. 6.1.24*:

♦ `(vlax-dump-object ax_circle T)`

В листинге 6.3 приведен результат распечатки свойств и методов объекта ax_circle, построенного с помощью русской версии AutoCAD 2005.

Листинг 6.3. Дамп свойств построенной окружности

```
; IAcadCircle: Интерфейс AutoCAD для работы с кругами.
; Значения свойств:
;   Application (RO) = #<VLA-OBJECT IAcadApplication 00b9b5e4>
;   Area = 31415.9
;   Center = (124.4 25.9 40.7)
;   Circumference = 628.319
;   Diameter = 200.0
;   Document (RO) = #<VLA-OBJECT IAcadDocument 0a08af90>
;   Handle (RO) = "EF"
;   HasExtensionDictionary (RO) = 0
;   Hyperlinks (RO) = #<VLA-OBJECT IAcadHyperlinks 0a0982c4>
;   Layer = "0"
;   Linetype = "ByLayer"
;   LinetypeScale = 1.0
;   Lineweight = -1
;   Normal = (0.0 0.0 1.0)
;   ObjectID (RO) = 2128236408
;   ObjectName (RO) = "AcDbCircle"
;   OwnerID (RO) = 2128235768
;   PlotStyleName = "ByLayer"
;   Radius = 100.0
;   Thickness = 0.0
;   TrueColor = #<VLA-OBJECT IAcadAcCmColor 0a098210>
;   Visible = -1
; Поддерживаемые методы:
;   ArrayPolar (3)
;   ArrayRectangular (6)
;   Copy ()
;   Delete ()
```

```
;   GetBoundingBox (2)
;   GetExtensionDictionary ()
;   GetXData (3)
;   Highlight (1)
;   IntersectWith (2)
;   Mirror (2)
;   Mirror3D (3)
;   Move (2)
;   Offset (1)
;   Rotate (2)
;   Rotate3D (3)
;   ScaleEntity (2)
;   SetXData (2)
;   TransformBy (1)
;   Update ()
```

6.1.26. Другие функции ActiveX

В расширение языка входит еще много полезных функций технологии ActiveX. Их можно было бы разделить на такие условные группы:

- функции команд и методов;
- функции операций с объектами;
- функции операций со словарями;
- функции преобразований;
- функции операций со свойствами;
- функции операций с кривыми.

Эти функции коротко рассматриваются в *разд. 6.2—6.7*.

Примеры программ, использующих технологию ActiveX, можно найти в папке Sample программного обеспечения системы AutoCAD.

6.2. Функции команд и методов

В следующих разделах рассматриваются четыре функции: `vlax-add-cmd`, `vlax-remove-cmd`, `vlax-invoke-method` и `vlax-product-key`.

6.2.1. Функция *vlax-add-cmd*

Позволяет создать команду AutoCAD через функцию, не имеющую префикса `c:`. Возможно даже создание прозрачных команд (такой возможности нет у функций с префиксом `c:`).

```
(vlax-add-cmd <глоб> <функция> [<лок> <флаги>])
```

Аргументы: <глоб> — строка с глобальным именем команды; <функция> — символ с именем функции, которая вызывается при обращении к команде (функция не должна иметь аргументов); <лок> — строка с локальным именем команды; <флаги> — сумма установленных битовых флагов (см. ниже). Если локальное имя команды не задано, то используется значение параметра <глоб>.

Возвращаемое значение — значение аргумента <глоб>. При сбое возвращается nil.

Данная функция предоставляет доступ к макросу ObjectARX acedRegCmds, который формирует указатель объекта AcEdCommandStack, содержащего имена команд.

Команды, создаваемые функцией vlax-add-cmd, заносятся в соответствующую группу команд (команды документа или команды VLX-приложения с отдельным именным пространством).

Не разрешается в качестве второго аргумента указывать символы функций, использующих реакторы или являющихся исполняемыми функциями реакторов *(см. главу 9)*.

Аргумент <флаги> формируется как сумма битовых флагов, представленных в табл. 6.4. Если какой-то флаг не установлен, то его значение считается равным нулю. По умолчанию для аргумента <флаги> принимается значение 4.

Таблица 6.4. *Битовые флаги, используемые функцией* `vlax-add-cmd`

Битовый флаг	Условное обозначение	Описание
1	ACRX_CMD_TRANSPARENT	Команда может выполняться в прозрачном режиме, т. е. до окончания предыдущей команды AutoCAD
2	ACRX_CMD_USEPICKSET	Команда может использовать набор предварительно выбранных объектов
4	ACRX_CMD_REDRAW	Команда может использовать как набор предварительно выбранных объектов, так и набор объектов, выбранных после ввода имени команды

Пример:

- `(defun mm () (alert "Совсем новая команда"))` создает простую функцию mm

  `(vlax-add-cmd "MY1" 'mm)` создает команду и возвращает "MY1"

При обращении к команде MY1 система откроет окно функции alert.

6.2.2. Функция *vlax-remove-cmd*

Удаляет команду или группу команд из списка команд в данном сеансе работы AutoCAD.

```
(vlax-remove-cmd <глоб>)
```

Аргумент — строка или `T`. Если значение аргумента — строка, то удаляется команда с данным именем. Если в качестве значения аргумента задано `T`, то удаляются команды, созданные данным приложением.

Возвращаемое значение — `T` (при успешном завершении) или `nil` (при невозможности выполнить удаление).

Пример:

- `(vlax-remove-cmd "MY1")` возвращает `T`

6.2.3. Функция *vlax-invoke-method*

Применяет метод с указанным именем.

```
(vlax-invoke-method <объект> <метод> <арг1> [<арг2> ... [<аргN>]...]])
```

Аргументы: `<объект>` — VLA-объект (не обязательно модели AutoCAD); `<метод>` — символ или строка с именем метода; `<арг1>`, `<арг2>`, ... `<аргN>` — значения аргументов, передаваемых методу (их количество определяется методом).

Возвращаемое значение определяется вызываемым методом.

Функция не проверяет значения, передаваемые в качестве значений аргументов `<арг1>`, `<арг2>`, ... `<аргN>`.

Замечание

В ранних версиях системы AutoCAD использовалась функция `vlax-invoke`.

Примеры:

- `(vlax-invoke-method model_space 'AddCircle center 100.0)` возвращает `#<VLA-OBJECT IAcadCircle 0a0a7354>`
- `(vlax-invoke-method model_space "AddCircle" center 100.0)` возвращает то же

6.2.4. Функция *vlax-product-key*

Вычисляет путь к системе AutoCAD в системном реестре Windows.

```
(vlax-product-key)
```

Возвращаемое значение — строка с путем, например (для русской версии AutoCAD 2005): `"Software\\Autodesk\\AutoCAD\\R16.1\\ACAD-301:419"`.

6.3. Функции операций с объектами

В следующих разделах изучаются функции, выполняющие различные операции с VLA-объектами. Некоторые аналогичные функции уже были рассмотрены в *разд. 6.1.4—6.1.7, 6.1.25*.

6.3.1. Функция *vlax-erased-p*

Проверяет, удален ли объект.

```
(vlax-erased-p <объект>)
```

Аргумент `<объект>` — VLA-объект.

Возвращаемое значение — `T`, если объект удален, и `nil`, если не удален.

6.3.2. Функция *vlax-method-applicable-p*

Проверяет, применим ли метод к данному объекту.

```
(vlax-method-applicable-p <объект> <метод>)
```

Аргументы: `<объект>` — VLA-объект; `<метод>` — символ (с апострофом) или строка с именем метода.

Возвращаемое значение — `T`, если метод применим к объекту `<объект>`, и `nil`, если не применим.

Примеры (переменные `model_space` и `acad_application` взяты из листинга 6.1):

- `(vlax-method-applicable-p model_space 'AddCircle)` возвращает `T`
- `(vlax-method-applicable-p acad_application "AddCircle")` возвращает `nil`

6.3.3. Функция *vlax-typeinfo-available-p*

Проверяет, присутствует ли в используемой библиотеке типов информация для объекта данного типа.

```
(vlax-typeinfo-available-p <объект>)
```

Аргумент `<объект>` — VLA-объект.

Возвращаемое значение — `T` (присутствует) или `nil` (не присутствует).

Прежде чем применить к объекту свойство или метод, система AutoCAD должна проверить библиотеку типов.

- `(vlax-typeinfo-applicable-p model_space)` возвращает `T`
- `(vlax-typeinfo-applicable-p acad_application)` возвращает `T`

6.3.4. Функция *vlax-object-released-p*

Проверяет, освобожден ли объект (т. е. освобождена ли выделенная для него память). Создаваемые VLA-объекты желательно освобождать. Очень важно использовать функцию в COM-технологии для работы с другими приложениями (Excel, Word и т. д.).

```
(vlax-object-released-p <объект>)
```

Аргумент `<объект>` — VLA-объект.

Удаленные объекты в течение некоторого времени не освобождаются, т. е. остаются доступными для восстановления. Восстановление оказывается невозможным либо в результате операции сборки мусора *(см. разд. 1.17.5)*, либо в результате применения функции `vlax-release-object` *(см. разд. 6.3.5)*, либо в результате закрытия сеанса AutoCAD.

Возвращаемое значение — `T`, если объект освобожден, и `nil` — в противном случае.

6.3.5. Функция *vlax-release-object*

Освобождает объект (об освобождении *см. разд. 6.3.4*). Важная функция при использовании VLA-объектов и COM-технологии.

```
(vlax-release-object <объект>)
```

Аргумент `<объект>` — VLA-объект.

Возвращаемое значение не определено, но по опыту применения при нормальном высвобождении обычно возвращается `0`.

6.3.6. Функция *vlax-read-enabled-p*

Проверяет, может ли объект быть прочитан.

```
(vlax-read-enabled-p <объект>)
```

Аргумент `<объект>` — VLA-объект.

Возвращаемое значение — `T` (может быть прочитан) или `nil` (не может быть прочитан).

6.3.7. Функция *vlax-write-enabled-p*

Определяет, может ли объект быть модифицирован (не открыт ли он только для чтения).

```
(vlax-write-enabled-p <объект>)
```

Аргумент `<объект>` — VLA-объект или имя примитива.

Возвращаемое значение — `T` (может быть модифицирован) или `nil` (не может быть модифицирован).

6.3.8. Функция *vlax-create-object*

Создает указатель на объект внешнего приложения перед его запуском.

```
(vlax-create-object <имя>)
```

Аргумент <имя> — строка, определяющая тип объекта. Может быть одного из следующих видов: "<приложение>.<тип>.<версия>" или "<приложение>.<тип>". Например: "Word.Application.11", "Excel.Application". Если версия не указана, то ее значение по умолчанию выбирается системой с помощью реестра Windows.

Возвращаемое значение — VLA-объект.

Если требуемый указатель (VLA-объект) ранее уже был создан и зарегистрирован как единичный объект, то повторно он не создается. *О работе с внешними приложениями см. разд. 10.4.*

6.3.9. Функция *vlax-get-object*

Вычисляет указатель (VLA-объект) для объекта внешнего приложения.

```
(vlax-get-object <имя>)
```

Аргумент <имя> — строка, определяющая тип объекта (см. описание аргумента функции vlax-create-object).

Возвращаемое значение — VLA-объект (если приложение запущено). При ошибке (если приложение еще не запущено или уже закрыто) возвращается nil.

6.3.10. Функция *vlax-get-or-create-object*

Вычисляет указатель объекта внешнего приложения или создает его, если такого указателя не было (т. е. приложение еще не стартовало).

```
(vlax-get-or-create-object <имя>)
```

Аргумент <имя> — строка, определяющая приложение (см. описание аргумента функции vlax-create-object).

Возвращаемое значение — VLA-объект.

6.3.11. Функция *vlax-import-type-library*

Импортирует информацию из библиотеки типов внешнего приложения. После импорта становятся доступными специализированные функции для работы со свойствами, методами и константами объектной модели этого приложения.

```
(vlax-import-type-library :tlb-filename <файл> [:methods-prefix <мпрефикс>
:properties-prefix <свпрефикс> :constants-prefix <конпрефикс>])
```

Аргумент <файл> может быть строкой с именем файла одного из следующих типов:

- библиотека типов (файл с расширением tlb);
- библиотека объектов (файл с расширением olb);
- исполняемый файл (с расширением exe);

- библиотека с ресурсом библиотеки типов (файл с расширением dll);
- сложный документ, содержащий библиотеку типов;
- файл другого формата, откуда может быть извлечена информация библиотеки типов.

Если в аргументе <файл> путь не приведен, то файл разыскивается по стандартному алгоритму поиска файлов поддержки.

Аргумент <мпрефикс> — строка, которая будет добавляться при импорте к формируемым именам функций, соответствующих методам. По умолчанию имеет значение пустой строки.

Аргумент <свпрефикс> — строка, которая будет добавляться к формируемым именам функций чтения и модификации свойств (например, для префикса xxx- и свойства Marvel генерируются имена xxx-get-marvel и xxx-put-marvel). По умолчанию имеет значение пустой строки.

Аргумент <конпрефикс> — строка, которая будет добавляться при импорте к формируемым именам констант (перечислений). По умолчанию имеет значение пустой строки.

Возвращаемое значение — T при успешном завершении, nil — при ошибке.

Если библиотеку типов внешнего приложения не загружать, то сохраняется способ доступа к методам и свойствам этого приложения с помощью функций vlax-invoke-method, vlax-get-property, vlax-put-property.

6.3.12. Функция *vlax-map-collection*

Применяет функцию сразу ко всем объектам семейства.

```
(vlax-map-collection <объект> '<функция>)
```

Аргументы: <объект> — VLA-объект, соответствующий семейству; <функция> — символ с именем функции или LISP-выражение, описывающее непоименованную функцию (см. описание функции lambda, *разд. 1.16.1*).

Возвращаемое значение — значение аргумента <объект>.

6.3.13. Функция *vlax-for*

Вычисляет выражения со всеми объектами семейства.

```
(vlax-for <символ> <семейство> [<выражение1> [<выражение2> ...
[<выражениеN>]...]])
```

Аргументы: <символ> — символ, вместо которого в выражения нужно поочередно подставлять объекты семейства; <семейство> — VLA-объект, соответствующий семейству, к объектам которого применяются выражения; <выражение1>, <выражение2>, ... <выражениеN> — LISP-выражения, которые нужно вычислить.

Возвращаемое значение — значение последнего выражения, вычисленное с последним объектом семейства.

6.4. Функции операций со словарями

Пять приведенных в следующих разделах функций выполняют операции с пользовательскими данными, сохраняемыми в словарях. Как было сказано в *разд. 1.15*, словари могут быть привязаны не только к документу (рисунку), но и к любому объекту (графическому и неграфическому) или семейству. Значения, записываемые в словарь, хранятся в виде пары: имя (ключ) и непосредственно данные. Поиск в словаре производится по ключевому имени, которое должно быть строкой и не должно повторяться внутри одного словаря. Хранение данных организовано в системе с помощью X-записей.

6.4.1. Функция *vlax-ldata-put*

Записывает LISP-данные в словарь.

```
(vlax-ldata-put <словарь> <имя> <данное> [<приложение>])
```

Аргументы: `<словарь>` — VLA-объект, соответствующий объекту рисунка, или строка с именем словаря; `<имя>` — ключевое имя элемента словаря; `<данное>` — данное любого типа, используемого в языке AutoLISP (число, список, строка и т. п.); `<приложение>` — если не опущен и не `nil`, то указывает на необходимость сохранения данных в отдельном именном пространстве выполняемого приложения.

Возвращаемое значение — значение аргумента `<данные>`.

Если в качестве первого аргумента указан VLA-объект, то данные заносятся в его словарь (Extension Dictionary), если строка — то в словарь с таким именем внутри рисунка (такой словарь является элементом корневого словаря). Если требуемый словарь еще не существует, то функция `vla-ldata-put` его сразу и создает.

Приложение может сохранять часть своих данных в рисунке, а часть — в своем пространстве.

Пример (в английской версии AutoCAD 2006):

- `(vlax-ldata-put "BOOK_DICT" "возраст" 17.6)` создает в рисунке словарь BOOK_DICT (добавляя его имя в корневой словарь), в словаре создает элемент с ключом `"возраст"`, записывает в этот элемент число и возвращает `17.6`

  `(entget (namedobj))` для нового рисунка возвращает

```
((-1 . <Entity name: 7ef65c60>) (0 . "DICTIONARY") (330 . <Entity name: 0>)
(5 . "C") (100 . "AcDbDictionary") (280 . 0) (281 . 1)
(3 . "ACAD_COLOR") (350 . <Entity name: 7ef65ce0>)
(3 . "ACAD_GROUP") (350 . <Entity name: 7ef65c68>)
(3 . "ACAD_LAYOUT") (350 . <Entity name: 7ef65cd0>)
```

```
(3 . "ACAD_MATERIAL") (350 . <Entity name: 7ef65cd8>)
(3 . "ACAD_MLINESTYLE") (350 . <Entity name: 7ef65cb8>)
(3 . "ACAD_PLOTSETTINGS") (350 . <Entity name: 7ef65cc8>)
(3 . "ACAD_PLOTSTYLENAME") (350 . <Entity name: 7ef65c70>)
(3 . "ACAD_TABLESTYLE") (350 . <Entity name: 7ef65ce8>)
(3 . "BOOK_DICT") (350 . <Entity name: 7ef65d78>))
```

В последнем списке видно имя нового словаря.

6.4.2. Функция *vlax-ldata-get*

Извлекает LISP-данные из словаря рисунка или из объекта.

```
(vlax-ldata-get <словарь> <имя> [<умолчание> [<приложение>]])
```

Аргументы: `<словарь>` — VLA-объект, соответствующий объекту рисунка, или строка с именем словаря; `<имя>` — ключевое имя элемента словаря; `<умолчание>` — значение любого типа, возвращаемое в том случае, если элемента с нужным именем в словаре нет (можно в качестве значения этого аргумента задавать `nil`); `<приложение>` — если не опущен и не `nil`, то указывает на необходимость извлечения данных из отдельного именного пространства выполняемого приложения.

Возвращаемое значение — прочитанное данное.

Пример (продолжение примера из *разд. 6.4.2*):

- `(vlax-ldata-get "BOOK_DICT" "возраст")` возвращает `17.6`

6.4.3. Функция *vlax-ldata-list*

Выводит список из точечных пар с кодами и значениями элементов словаря.

```
(vlax-ldata-list <словарь> [<приложение>])
```

Аргументы: `<словарь>` — VLA-объект, соответствующий графическому объекту рисунка, или строка с именем словаря; `<приложение>` — если не опущен и не `nil`, то указывает на необходимость извлечения данных из отдельного именного пространства выполняемого приложения.

Возвращаемое значение — список из точечных пар (первый элемент пары — ключ, а второй элемент — данное) типа `(("tank" . "T-34") ("bank" . '(57))).`

Пример (продолжение примера из *разд. 6.4.2*):

- `(vlax-ldata-list "BOOK_DICT")` возвращает `(("возраст" . 17.6))`

6.4.4. Функция *vlax-ldata-delete*

Удаляет LISP-данные из словаря.

```
(vlax-ldata-delete <словарь> <имя> [<приложение>])
```

Аргументы: <словарь> — VLA-объект, соответствующий объекту рисунка, или строка с именем словаря; <имя> — ключевое имя элемента словаря; <приложение> — если не опущен и не nil, то указывает на необходимость удаления данных из пространства имен выполняемого приложения.

Возвращаемое значение — T при успешном выполнении удаления или nil — при невозможности удаления.

6.4.5. Функция *vlax-ldata-test*

Проверяет, могут ли быть сохранены данные по окончании сеанса работы AutoCAD.

```
(vlax-ldata-test <данные>)
```

Аргумент <данные> — LISP-данное, которое проверяется на возможность сохранения.

Возвращаемое значение — T (данные могут быть сохранены) или nil (данные не могут быть сохранены).

6.4.6. Словарь Extension Dictionary

С каждым объектом базы рисунка можно связать словарь дополнительных данных, аналогично расширенным данным. Такой словарь называется *словарем расширения объекта* (extension dictionary). В таких словарях хранят некоторые свои данные как AutoCAD, так и многие его специализированные приложения-надстройки.

Как выяснить, имеет ли VLA-объект vlo3 словарь расширения? Для этого необходимо прочитать значение свойства HasExtensionDictionary объекта vlo3:

```
(setq hased (vla-get-HasExtensionDictionary vlo3))
```

Выражение возвратит :vlax-true (да) или :vlax-false (нет) в зависимости от наличия такого словаря. Далее с помощью метода GetExtensionDictionary можно получить VLA-объект этого словаря и затем поработать с его данными. В словарь могут быть вложены другие словари, которые позволяют создать структуру любой сложности.

6.4.6.1. Словарь фильтров слоев

В листинге 6.4 приведен пример доступа к фильтрам слоев, которые располагаются в словаре, имеющемся у семейства Layers (семейство слоев). В примере текущий рисунок имеет один групповой фильтр слоев и один фильтр слоев по свойствам.

Листинг 6.4. Доступ к фильтрам слоев

```
; Глава 06\Book02\lfilters.lsp
(vl-load-com)
; Активный документ (VLA-объект)
(setq actdoc (vla-get-ActiveDocument (vlax-get-acad-object)))
; Семейство слоев (VLA-объект)
(setq ls (vla-get-Layers actdoc))
; Проверка наличия признака словаря расширения
(setq hased (vla-get-HasExtensionDictionary ls)); :vlax-true
; Получение VLA-объекта словаря расширения
(setq edic (vla-getExtensionDictionary ls))

; Получение примитива словаря (для использования функции entget)
(setq dl (entget (vlax-vla-object->ename edic)))
;((-1 . <Entity name: 7ef65dd0>) (0 . "DICTIONARY")
;(330 . <Entity name: 7ef65c10>) (5 . "3A")
;(100 . "AcDbDictionary") (280 . 1) (281 . 1)
;(3 . "ACAD_LAYERFILTERS") (360 . <Entity name: 7ef65eb0>)
;(3 . "ACLYDICTIONARY") (360 . <Entity name: 7ef65ea8>))

; Выделение имен вложенных словарей (по DXF-коду 3)
(setq dfnames (mapcar 'cdr
  (vl-remove-if-not (function (lambda (x) (= (car x) 3))) dl)))
;("ACAD_LAYERFILTERS" "ACLYDICTIONARY")

; Выделение примитива первого словаря (ACAD_LAYERFILTERS)
; Это словарь групповых фильтров слоев
(setq d01 (cdadr (member (cons 3 (nth 0 dfnames)) dl)))
(setq ld01 (entget d01))
;((-1 . <Entity name: 7ef65eb0>) (0 . "DICTIONARY") (5 . "56")
;(102 . "{ACAD_REACTORS") (330 . <Entity name: 7ef65dd0>)
;(102 . "}") (330 . <Entity name: 7ef65dd0>)
;(100 . "AcDbDictionary") (280 . 0) (281 . 1)
;(3 . "RedF") (350 . <Entity name: 7ef65ec0>))

; Выделение примитива второго словаря (ACLYDICTIONARY)
; Это словарь фильтров слоев по свойствам
(setq d02 (cdadr (member (cons 3 (nth 1 dfnames)) dl)))
(setq ld02 (entget d02))
;((-1 . <Entity name: 7ef65ea8>) (0 . "DICTIONARY") (5 . "55")
;(102 . "{ACAD_REACTORS") (330 . <Entity name: 7ef65dd0>)
;(102 . "}") (330 . <Entity name: 7ef65dd0>)
;(100 . "AcDbDictionary") (280 . 0) (281 . 1)
;(3 . "*A1") (350 . <Entity name: 7ef65eb8>)
;(3 . "*A2") (350 . <Entity name: 7ef65ec8>))
```

В первых строках этого листинга последовательно вычисляются:

- actdoc — VLA-объект текущего документа;
- ls — VLA-объект семейства слоев;
- hased — признак наличия у семейства слоев словаря (это :vlax-true);
- edic — VLA-объект словаря.

Далее с помощью выражения (setq dl (entget (vlax-vla-object->ename edic))), в котором функция vlax-vla-object->ename преобразует VLA-объект в примитив, получаем список dl с данными словаря как примитива: ((-1 . <Entity name: 7ef65dd0>) (0 . "DICTIONARY");(330 . <Entity name: 7ef65c10>) (5 . "3A") (100 . "AcDbDictionary") (280 . 1) (281 . 1) (3 . "ACAD_LAYERFILTERS") (360 . <Entity name: 7ef65eb0>) (3 . "ACLYDICTIONARY") (360 . <Entity name: 7ef65ea8>)).

Анализ этого списка показывает, что словарь edic содержит два элемента, ключевые имена которых "ACAD_LAYERFILTERS" и "ACLYDICTIONARY" (в листинге показано, как извлечь из списка dl имена элементов в список dfnames).

В качестве значений элементов словаря edic выступают примитивы (они записаны в точечных парах с DXF-кодом 360). Дальнейший анализ показывает, что этими объектами являются словари — первый из них является словарем групповых фильтров слоев, а второй — словарем фильтров слоев по свойствам.

В последних строках листинга получаются списки с данными обоих этих словарей. Если проанализировать списки, то можно увидеть, что имя группового фильтра — RedF, а фильтров по свойствам два (они обозначены условными именами *A1 и *A2). Аналогичный разбор можно продолжить дальше, получив конкретные свойства обнаруженных фильтров.

6.4.6.2. Объект VLO-VL

Примерно также размещаются в словаре данные с помощью функции vlax-ldata-put, которую удобно использовать для записи в примитив простых данных пользователя. Однако если функцией entget вывести список с DXF-кодами примитива, который создается системой для хранения данных в случае применения vlax-ldata-put, то увидим результаты, приведенные в листинге 6.5.

Листинг 6.5. Данные, формируемые с помощью функции vlax-ldata-put

```
; Глава 06\Book02\ldata.lsp
; Выбор объекта для хранения своих данных
(setq ve (vlax-ename->vla-object (car (entsel))))
; Запись данных в элемент "AAA" словаря объекта
(vlax-ldata-put ve "AAA" 1000)
; Запись данных в элемент "BBB" словаря объекта
(vlax-ldata-put ve "BBB" "данные для BBB")
```

```
; Словарь расширения объекта
(setq exd (vla-getExtensionDictionary ve))
(setq led (entget (vlax-vla-object->ename exd)))
;((-1 . <Entity name: 7efa2f70>) (0 . "DICTIONARY")
;(330 . <Entity name: 7efa2f68>) (5 . "E6")
;(100 . "AcDbDictionary") (280 . 1) (281 . 1)
;(3 . "AAA") (360 . <Entity name: 7efa2f78>)
;(3 . "BBB") (360 . <Entity name: 7efa2f88>))

; Доступ к X-записи с данными элемента "AAA"
(setq xr1 (cdadr (member '(3 . "AAA") led)))
(entget xr1)
;((-1 . <Entity name: 7efa2f78>) (0 . "VLO-VL") (5 . "E7")
;(102 . "{ACAD_REACTORS") (330 . <Entity name: 7efa2f70>)
;(102 . "}") (330 . <Entity name: 7efa2f70>)
;(100 . "vlo_VL") (90 . -64512) (91 . 4) (92 . 0)
;(300 . "1000"))

; Доступ к X-записи с данными элемента "BBB"
(setq xr2 (cdadr (member '(3 . "BBB") led)))
(entget xr2)
;((-1 . <Entity name: 7efa2f88>) (0 . "VLO-VL") (5 . "E9")
;(102 . "{ACAD_REACTORS") (330 . <Entity name: 7efa2f70>)
;(102 . "}") (330 . <Entity name: 7efa2f70>)
;(100 . "vlo_VL") (90 . -64512) (91 . 16) (92 . 0)
;(300 . "\"данные для BBB\""))
```

В листинге 6.5 сначала выбирается графический объект, вместе с которым мы будем хранить свои данные, и вычисляем соответствующий ему VLA-объект `ve`.

Затем с помощью функции `vlax-ldata-put` заносим в словарь расширения объекта данные: в элемент с ключевым именем `"AAA"` — число `1000`, а в элемент с именем `"BBB"` — строку `"данные для BBB"`.

Доступ к элементам словаря осуществляется через список данных словаря `led`. Как видно из листинга, в этом списке точечные пары с DXF-кодом 3 содержат имена элементов (`"AAA"`, `"BBB"`), точечные пары с DXF-кодом 360 — соответствующие примитивы базы рисунка, хранящие данные пользователя (`<Entity name: 7efa2f78>`, `<Entity name: 7efa2f88>`). Эти примитивы обозначены переменными `xr1` и `xr2`.

Самая интересная информация получится, если мы к примитивам `xr1` и `xr2` применим функцию `entget` и рассмотрим возвращаемые списки. Оказывается, эти примитивы имеют не известный доселе тип — VLO-VL!

Недокументированные объекты VLO-VL являются X-записями, приспособленными под нужды функций с префиксом `vlax-ldata-`. Сами данные элементов

словаря хранятся в точечных парах с кодом 300, а в коде 91 содержатся сведения о типе данных (для правильности извлечения).

6.5. Функции преобразований

Несколько функций, оперирующих безопасными массивами, уже рассмотрены в *разд. 6.1*. В следующих разделах приводятся еще шесть функций, а также функция `vlax-tmatrix`.

6.5.1. Функция *vlax-safearray-get-dim*

Вычисляет размерность безопасного массива.

```
(vlax-safearray-get-dim <бм>)
```

Аргумент `<бм>` — безопасный массив.

Возвращаемое значение — целое число (размерность массива). Если аргумент `<бм>` — не безопасный массив, то выдается сообщение об ошибке.

6.5.2. Функция *vlax-safearray-get-l-bound*

Вычисляет начальное (нижнее) значение индекса безопасного массива.

```
(vlax-safearray-get-l-bound <бм> <номер>)
```

Аргументы: `<бм>` — безопасный массив; `<номер>` — номер индекса, для которого вычисляется начальное значение (индекс первого измерения массива имеет номер 1, второго — 2 и т. д.).

Возвращаемое значение — целое число с начальным значением индекса. Если переданы неверные данные в первом или втором аргументах, то выдается сообщение об ошибке.

6.5.3. Функция *vlax-safearray-get-u-bound*

Вычисляет конечное (верхнее) значение индекса безопасного массива.

```
(vlax-safearray-get-u-bound <бм> <номер>)
```

Аргументы: `<бм>` — безопасный массив; `<номер>` — номер индекса, для которого вычисляется конечное значение (индекс первого измерения массива имеет номер 1, второго — 2 и т. д.).

Возвращаемое значение — целое число с конечным значением индекса. Если переданы неверные данные в первом или втором аргументах, то выдается сообщение об ошибке.

6.5.4. Функция *vlax-safearray-get-element*

Вычисляет значение элемента безопасного массива по индексам.

```
(vlax-safearray-get-element <бм> <инд1> [<инд2> ... [<индN>]...])
```

Аргументы: <бм> — безопасный массив; <инд1>, <инд2>, … <индN> — целые числа (значения индексов, для которых вычисляется значение элемента). Количество аргументов с индексами должно совпадать с размерностью массива (для одномерного массива это одно число).

Возвращаемое значение — значение элемента массива.

6.5.5. Функция *vlax-safearray-type*

Вычисляет тип безопасного массива.

```
(vlax-safearray-type <бм>)
```

Аргумент <бм> — безопасный массив.

Возвращаемое значение — целое число (2, 3, 4, 5, 8, 9 или 11), определяющее тип массива в соответствии с табл. 6.2. Для массива из вариантов возвращается 12. Если в качестве аргумента передан не безопасный массив, то выдается сообщение об ошибке.

6.5.6. Функция *vlax-safearray->list*

Преобразует элементы массива в список.

```
(vlax-safearray->list <бм>)
```

Аргумент <бм> — безопасный массив.

Возвращаемое значение — список, составленный из элементов массива.

6.5.7. Функция *vlax-tmatrix*

Вычисляет вариант типа массива (`vlax-vbArray`) для методов, выполняющих операции, аналогичные преобразованию с матрицей 4×4.

```
(vlax-tmatrix <список>)
```

Аргумент <список> — список, элементами которого являются четыре списка из четырех вещественных чисел. Этот список представляет собой матрицу размером 4×4.

Возвращаемое значение — вариант, представляющий переданную матрицу преобразования.

6.6. Функции операций со свойствами

Помимо работающих со свойствами функций, рассмотренных в *разд. 6.1*, в расширение языка AutoLISP входят следующие: `vlax-get-property`, `vlax-put-property`, `vlax-property-available-p`.

6.6.1. Функция *vlax-get-property*

Вычисляет свойство объекта.

```
(vlax-get-property <объект> <свойство>)
```

Аргументы: `<объект>` — VLA-объект, `<свойство>` — символ (с апострофом) или строка с наименованием свойства.

Возвращаемое значение — значение свойства (тип значения зависит от свойства).

Замечание

В ранних версиях системы AutoCAD использовалась функция `vlax-get`.

6.6.2. Функция *vlax-put-property*

Присваивает новое значение свойству объекта.

```
(vlax-put-property <объект> <свойство> <значение>)
```

Аргументы: `<объект>` — VLA-объект, `<свойство>` — символ (с апострофом) или строка с наименованием свойства; значение — новое значение свойства.

Возвращаемое значение — `nil`. При сбое выводится сообщение об ошибке.

Замечание

В ранних версиях системы AutoCAD использовалась функция `vlax-put`.

6.6.3. Функция *vlax-property-available-p*

Проверяет, имеет ли объект данное свойство.

```
(vlax-property-available-p <объект> <свойство> [<модификация>])
```

Аргументы: `<объект>` — VLA-объект; `<свойство>` — символ (с апострофом) или строка с именем свойства; `<модификация>` — признак проверки возможности модификации свойства (если `T`, то такая проверка выполняется).

Возвращаемое значение — `T` или `nil`, зависит от значения третьего аргумента:

1. Аргумент `<модификация>` не задан или имеет значение `nil`, то возвращается `T`, если данное свойство у объекта есть, `nil` — если данного свойства нет.

2. Аргумент <модификация> задан и имеет значение T, то возвращается T, если данное свойство у объекта есть и возможно его изменение, nil — если данного свойства нет или его нельзя модифицировать.

6.7. Функции операций с кривыми

Расширение AutoLISP предоставляет пользователю большое количество функций для дополнительных операций с кривыми (в том числе операций доступа к параметрическому заданию кривых). К кривым относятся все простые примитивы AutoCAD (отрезок, луч, сплайн и т. д.).

6.7.1. Функция *vlax-curve-getArea*

Вычисляет площадь внутри кривой.

```
(vlax-curve-getArea <кривая>)
```

Аргумент <кривая> — примитив или VLA-объект, соответствующий кривой.

Возвращаемое значение — вещественное число. В случае сбоя возвращается nil.

6.7.2. Функция *vlax-curve-getStartPoint*

Вычисляет начальную точку кривой в МСК.

```
(vlax-curve-getStartPoint <кривая>)
```

Аргумент <кривая> — примитив или VLA-объект.

Возвращаемое значение — список из трех вещественных чисел (координат). В случае сбоя возвращается nil.

6.7.3. Функция *vlax-curve-getEndPoint*

Вычисляет конечную точку кривой в МСК.

```
(vlax-curve-getEndPoint <кривая>)
```

Аргумент <кривая> — примитив или VLA-объект.

Возвращаемое значение — список из трех вещественных чисел (координат). В случае сбоя возвращается nil.

6.7.4. Функция *vlax-curve-getPointAtDist*

Вычисляет точку (в МСК), которая лежит на кривой и отстоит по длине этой кривой от начальной точки на заданное расстояние.

```
(vlax-curve-getPointAtDist <кривая> <расстояние>)
```

Аргументы: <кривая> — примитив или VLA-объект, соответствующий кривой; <расстояние> — вещественное число, задающее расстояние от начальной точки кривой.

Возвращаемое значение — список из трех вещественных чисел (координат). В случае сбоя возвращается nil.

6.7.5. Функция *vlax-curve-getDistAtPoint*

Вычисляет длину кривой между начальной точкой и заданной точкой на кривой.

```
(vlax-curve-getDistAtPoint <кривая> <точка>)
```

Аргументы: <кривая> — примитив или VLA-объект; <точка> — список из трех вещественных чисел с координатами точки в МСК.

Возвращаемое значение — вещественное число. В случае ошибки возвращается nil.

6.7.6. Функция *vlax-curve-getStartParam*

Вычисляет значение параметра в начальной точке кривой.

```
(vlax-curve-getStartParam <кривая>)
```

Аргумент <кривая> — примитив или VLA-объект.

Возвращаемое значение — вещественное число. В случае сбоя возвращается nil.

6.7.7. Функция *vlax-curve-getEndParam*

Вычисляет значение параметра в конечной точке кривой.

```
(vlax-curve-getEndParam <кривая>)
```

Аргумент <кривая> — примитив или VLA-объект.

Возвращаемое значение — вещественное число. В случае сбоя возвращается nil.

6.7.8. Функция *vlax-curve-getParamAtPoint*

Вычисляет значение параметра в заданной точке кривой.

```
(vlax-curve-getParamAtPoint <кривая> <точка>)
```

Аргументы: <кривая> — примитив или VLA-объект; <точка> — список из трех вещественных чисел с координатами точки в МСК.

Возвращаемое значение — вещественное число. В случае ошибки возвращается nil.

6.7.9. Функция *vlax-curve-getPointAtParam*

Вычисляет точку, лежащую на кривой и имеющую заданное значение параметра.

```
(vlax-curve-getPointAtParam <кривая> <параметр>)
```

Аргументы: `<кривая>` — примитив или VLA-объект; `<параметр>` — вещественное число (значение параметра).

Возвращаемое значение — список из трех вещественных чисел с координатами точки (в МСК). В случае ошибки возвращается `nil`.

6.7.10. Функция *vlax-curve-getParamAtDist*

Вычисляет значение параметра для точки, лежащей на кривой и отстоящей по длине этой кривой от начальной точки на заданное расстояние.

```
(vlax-curve-getParamAtDist <кривая> <расстояние>)
```

Аргументы: `<кривая>` — VLA-объект или примитив; `<расстояние>` — вещественное число.

Возвращаемое значение — вещественное число. В случае ошибки возвращается `nil`.

6.7.11. Функция *vlax-curve-getDistAtParam*

Вычисляет длину сегмента кривой от начальной точки до точки с заданным значением параметра.

```
(vlax-curve-getDistAtParam <кривая> <параметр>)
```

Аргументы: `<кривая>` — примитив или VLA-объект; `<параметр>` — вещественное число.

Возвращаемое значение — вещественное число. В случае ошибки возвращается `nil`.

6.7.12. Функция *vlax-curve-isClosed*

Проверяет, является ли кривая замкнутой (т. е. совпадает ли конечная точка с начальной).

```
(vlax-curve-isClosed <кривая>)
```

Аргумент `<кривая>` — примитив или VLA-объект.

Возвращаемое значение — `T` (да) или `nil` (нет).

6.7.13. Функция *vlax-curve-isPeriodic*

Проверяет, является ли кривая периодической, т. е. параметр ее уравнения имеет бесконечный диапазон значений в обе стороны и точки на кривой, отличающиеся значениями параметра на некоторую величину `dT` (период), совпадают.

```
(vlax-curve-isPeriodic <кривая>)
```

Аргумент `<кривая>` — примитив или VLA-объект.

Возвращаемое значение — `T` (да) или `nil` (нет).

6.7.14. Функция *vlax-curve-isPlanar*

Проверяет, лежит ли кривая целиком в некоторой плоскости.

```
(vlax-curve-isPlanar <кривая>)
```

Аргумент `<кривая>` — примитив или VLA-объект.

Возвращаемое значение — `T` (да) или `nil` (нет).

6.7.15. Функция *vlax-curve-getClosestPointTo*

Вычисляет точку (в МСК), которая лежит на кривой и является ближайшей к заданной точке.

```
(vlax-curve-getClosestPointTo <кривая> <точка> [<продолжение>])
```

Аргументы: `<кривая>` — примитив или VLA-объект; `<точка>` — список из трех вещественных чисел с координатами точки в МСК; `<продолжение>` — признак (если задан и не `nil`, то точка ищется на продолжении кривой).

Возвращаемое значение — список из трех вещественных чисел с координатами точки (в МСК), ближайшей к заданной. В случае ошибки возвращается `nil`.

6.7.16. Функция *vlax-curve-getClosestPointToProjection*

Вычисляет точку (в МСК), которая лежит на кривой и при проектировании на указанную плоскость является ближайшей к заданной точке.

```
(vlax-curve-getClosestPointToProjection <кривая> <точка> <нормаль>
[<продолжение>])
```

Аргументы: `<кривая>` — примитив или VLA-объект; `<точка>` — список из трех вещественных чисел с координатами точки в МСК (через эту точку строится плоскость, на которую будет выполняться проектирование); `<нормаль>` — список из трех вещественных чисел с координатами точки (в МСК), определяющими положение нормали к плоскости проекции; `<продолжение>` — признак (если задан и не `nil`, то точка ищется на продолжении кривой).

Возвращаемое значение — список из трех вещественных чисел с координатами точки (в МСК), ближайшей к заданной точке при проектировании на плоскость. В случае ошибки возвращается `nil`.

6.7.17. Функция *vlax-curve-getFirstDeriv*

Вычисляет первую производную (в МСК) к кривой в точке, заданной параметром.

```
(vlax-curve-getFirstDeriv <кривая> <параметр>)
```

Аргументы: `<кривая>` — примитив или VLA-объект, соответствующий кривой; `<параметр>` — значение параметра, определяющее положение точки на кривой.

Возвращаемое значение — список из трех вещественных чисел (координаты вектора первой производной). В случае ошибки возвращается `nil`.

6.7.18. Функция *vlax-curve-getSecondDeriv*

Вычисляет вторую производную (в МСК) к кривой в точке, заданной параметром.

```
(vlax-curve-getSecondDeriv <кривая> <параметр>)
```

Аргументы: `<кривая>` — примитив или VLA-объект, соответствующий кривой; `<параметр>` — значение параметра, определяющее положение точки на кривой.

Возвращаемое значение — список из трех вещественных чисел (координаты вектора второй производной). В случае ошибки возвращается `nil`.

6.7.19. Что такое параметр кривой

Интерпретация параметра, связываемого с примитивом, зависит от типа примитива. Получить такую интерпретацию можно путем несложных экспериментов. Приведем результаты для некоторых объектов:

- отрезок — расстояние от начальной точки;
- дуга — центральный угол в радианах;
- окружность — центральный угол в радианах;
- полилиния — количество предыдущих участков от начала (целая часть параметра) и доля следующего участка (в дробной части); например, 1.5 — середина второго участка полилинии. Целые значения соответствуют вершинам.

6.8. Примеры работы с технологией ActiveX

В двух следующих главах приводится энциклопедическое описание методов и свойств объектов и семейств системы AutoCAD. В данном разделе покажем только два примера практического использования технологии ActiveX. См. также *приложение 5*.

6.8.1. Доступ к неактивному документу

Когда одновременно открыто несколько рисунков, то с помощью LISP-программы, запущенной в активном рисунке, можно получить доступ ко всем другим документам. Нельзя только активизировать другой документ, т. к. работа программы прервется.

Из предыдущего примера видно, как получить семейство открытых документов. Все семейства имеют метод `Item`, который предоставляет доступ к элементам по номеру или по имени. Нумерация элементов семейства начинается **с нуля**.

В листинге 6.6 приведен текст функции, которая для открытого документа с номером `N` возвращает имя его текущего слоя.

Листинг 6.6. Функция `anydoc_clayer`

```
; Глава 06\Book03\anydoc_clayer.lsp
(defun anydoc_clayer (N / _OpenDocs)
(vl-load-com)
; Семейство открытых документов
(setq _OpenDocs (vla-get-Documents (vlax-get-acad-object)))
; Проверка номера документа и возврат имени текущего слоя
(if (and (= 'INT (type N)) (>= N 0) (< N (vla-get-Count _OpenDocs)))
  (vla-get-name (vla-get-ActiveLayer (vla-item _OpenDocs N)))
);if
);defun anydoc_clayer
```

В этой функции проверяется тип аргумента `N` (он должен быть целочисленным) и правильность значения (оно должно быть неотрицательным и меньше количества элементов семейства, которое получается с помощью свойства `Count`). В случае ошибки программа возвращает `nil`, а в случае успеха — имя текущего слоя указанного документа.

6.8.2. Динамическое изменение контекстного меню

В системе AutoCAD падающие и контекстные меню образуют одно семейство `PopupMenus`, доступ к которому осуществляется с помощью свойства `Menus` объекта группы меню (адаптации). Обращение к нужному меню с помощью метода `Item` лучше выполнять по имени меню, а не по номеру в семействе (тем более что в разных версиях порядок меню разный, а имена обычно сохраняются).

Получение списка имен контекстных и падающих меню группы ACAD приведено в листинге 6.7.

Листинг 6.7. Функция menu_names

```
; Глава 06\Book03\menu_names.lsp
(defun menu_names ( / _mg _pms _m _pmnames)
(vl-load-com)
; Группа меню/адаптации ACAD
(setq _mg (vla-item (vla-get-MenuGroups (vlax-get-acad-object)) "ACAD"))
; Семейство падающих и контекстных меню
(setq _pms (vla-get-Menus _mg))
(vlax-for _m _pms (setq _pmnames (append _pmnames
  (list (vla-get-name _m)))))
);defun menu_names
```

В функции menu_names сначала вычисляется VLA-объект группы меню (адаптации) ACAD, а затем — семейство меню _pms падающих и контексных меню этой группы. Для английской версии AutoCAD 2006 функция возвращает следующий список: ("Attribute Block Objects Menu" "Block Reference Objects Menu" "Command Menu" "Default Menu" "Dimension Objects Menu" "Edit Menu" "&Grips Cursor Menu" "Hatch Object Menu" "LWPline Object Menu" "Maxmimized Viewport Menu" "Mtext Object Menu" "&Object Snap Cursor Menu" "&File" "&Edit" "&View" "&Insert" "F&ormat" "&Tools" "&Draw" "Dime&nsion" "&Modify" "&Window" "&Help" "Pline Object Menu" "Spline Object Menu" "Text Object Menu" "Unmaxmimized Viewport Menu" "Viewport Object Menu" "Viewport Objects Menu" "Xref Objects Menu" "Xref Object Menu"). В русской версии этот список выглядит так: ("Меню объектов блоков с атрибутами" "Меню объектов вхождения блока" "Меню команд" "Меню по умолчанию" "Меню объектов размеров" "Меню ''Правка''" "К&онтекстное меню ручек" "Меню объектов штриховки" "Меню объектов LWPline" "Меню развернутых видовых экранов" "Меню многострочного текста" "&Контекстное меню привязки" "&Файл" "&Правка" "&Вид" "Вст&авка" "Фор&мат" "&Сервис" "&Рисование" "Ра&змеры" "Ре&дакт" "&Окно" "Справ&ка" "Меню объектов полилиний" "Меню объектов сплайнов" "Меню текстового объекта" "Меню свернутых видовых экранов" "Меню объекта видовых экранов" "Меню объектов видовых экранов" "Меню объектов внешних ссылок" "Меню объекта внешних ссылок").

В качестве примера добавим в системе AutoCAD 2006 новый пункт в контекстное меню с названием **Default menu** (Меню по умолчанию)[3]. В листинге 6.8 приведен вариант программы для английской версии. Для русской версии следует использовать русское наименование меню.

Листинг 6.8. Функция item_shcmenu

```
; Глава 06\Book03\item_shcmenu.lsp
(defun item_shcmenu ( / _mg _pm _it)
(vl-load-com)
; Группа ACAD
(setq _mg (vla-item (vla-get-MenuGroups (vlax-get-acad-object)) "ACAD"))
```

[3] В версии 2005 это меню называется иначе: **Context menu for default mode** (Контекстное меню по умолчанию).

```
; Контекстное меню 2006 "Default menu"
; (в русской версии 2006 "Меню по умолчанию")
(setq _pm (vla-item (vla-get-Menus _mg) "Default menu"))
; Добавление нового пункта в начало контекстного меню
(setq _it (vla-AddMenuItem _pm 0 "Новый пункт"
  "\003\003\020(alert \"Вызов моего приложения: MYAP\")(princ) \020"))
; Если пункт добавлен, то формируем подсказку к нему
(if _it (vla-put-helpstring _it "Подсказка к новому пункту меню"))
(princ)
);defun item_shcmenu
```

В приведенной программе в начало контекстного меню (т. е. в позицию с номером 0) с помощью метода `AddMenuItem` добавляется пункт с названием `"Новый пункт"` и макросом `"\003\003\020(alert \"Вызов моего приложения: MYAP\")(princ) \020"`. В макросе вместо привычных символов `^C` и `^P` использованы соответствующие им восьмеричные числа \003 и \020. Макрос вызывает окно функции `alert` с сообщением. Функция `princ` в макросе и в конце программы применена для тихого выхода (без эхо-вывода в командной строке).

Добавление пункта осуществляется как добавление нового элемента в меню, которое является семейством пунктов. После создания пункта к нему с помощью изменения значения свойства `HelpString` присоединяется текст подсказки.

На рис. 6.1 показано контекстное меню английской версии AutoCAD 2006 с новым пунктом. Внизу видна подсказка.

Для удаления пункта из меню следует пользоваться методом `Delete`.

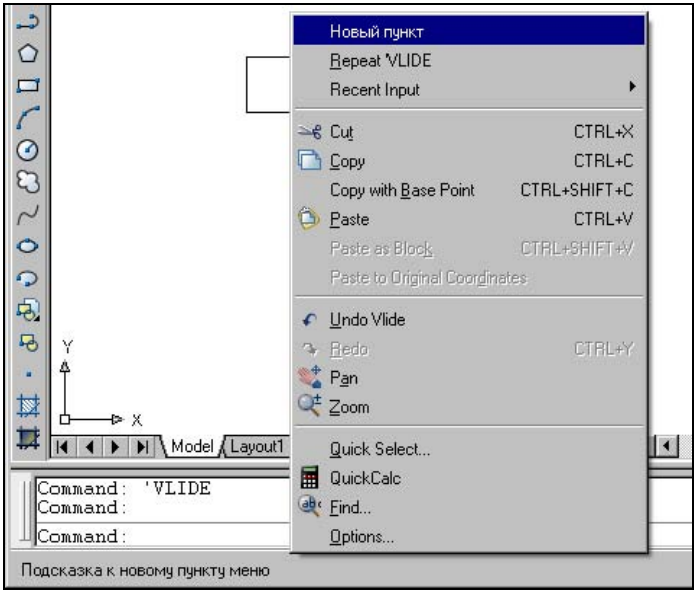
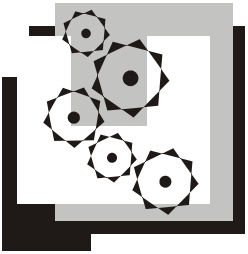


Рис. 6.1. Контекстное меню с добавленным пунктом

ГЛАВА 7



Методы ActiveX

В предыдущей главе был дан обзор объектной модели AutoCAD и описаны основные идеи ее использования. Данная глава посвящена подробному изложению функций Visual LISP, реализующих доступ к методам элементов объектной модели.

7.1. Дополнительные соглашения

Введем некоторые дополнительные соглашения. В примерах данной главы часто будет использоваться указатель на важный объект — активный документ. Будем считать, что у нас есть глобальная переменная **actdoc**, содержащая указатель на активный документ, т. е. перед выполнением примеров данной главы было выполнено следующее присвоение:

```
(setq actdoc (vla-get-ActiveDocument (vlax-get-acad-object)))
;;; #<VLA-OBJECT IAcadDocument 01043f00>
```

Естественно, что в разных сессиях работы указатель на объект документа будет разным, но для разбора методов это не играет роли. В строке комментария (она начинается с символов `;;;`) показан пример значения, возвращаемого системой (точнее, интерпретатором LISP).

Некоторые, но не все, функции, которые будут рассмотрены в этой главе, одинаково успешно могут применяться в контексте любого открытого документа, а не только активного. Привязка к активному документу делается для простоты воспроизведения читателем примеров, особенно тех, где манипуляции производятся с графическими объектами и в ходе примера меняется свойство, отображаемое на экране, а также ввиду того, что часть функций не могут быть успешно выполнены вне активного документа.

Для того чтобы читатель мог видеть, с какими конкретно объектами производятся манипуляции, мы вводим еще две переменные, которые будут использоваться на протяжении всей этой главы. В переменную `obj` мы будем помещать указатель на объект, с которым производятся манипуляции, а в переменную `temp` — данные, которые будут востребованы в ходе выполнения примера. Содержимое последней переменной может меняться динамически. Кроме того, в отдельных примерах фигурируют и другие переменные, назначение которых понятно по контексту выполнения. Переменные выделены в примерах полужирным начертанием.

Методы, появившиеся в версиях 2005 и 2006, помечены справа на поле книги соответствующим графическим изображением с номером версии.

7.2. Функции методов

Обращение к методам объектов через интерфейс ActiveX возможно тремя способами: либо через функцию `vlax-invoke-method`, либо через семейство именных функций методов, либо с использованием функции `vlax-invoke`. Названия всех именных функций AutoLISP этого типа формируются по единому правилу следующим образом: **`vla-<метод>`**, где `vla-` — префикс, `<метод>` — имя метода. Вызов именных функций, обращающихся к методам, производится в следующем формате:

```
(vla-<метод> <VLA-объект> [<аргументы>])
```

Следует учитывать, что обращение к методам объектов вышеупомянутыми способами не полностью эквивалентно. Функция `vlax-invoke-method` более универсальна и позволяет обращаться к методам объектов не только AutoCAD, но и любых других приложений, поддерживающих технологию ActiveX (т. е. доступ к своей объектной модели), **без импорта соответствующих библиотек типов**. Естественно, для этого необходимо знать имена методов объектов, к которым производится обращение, и правила их использования.

Функция `vlax-invoke` формально не документирована. Поэтому при ее использовании нельзя быть ни в чем уверенными. В том числе и в том, что в очередной версии AutoCAD она не окажется изъятой. Но, т. к. в последних версиях (2002—2006) она есть, то ее можно применять наравне с другими способами. Использование этой функции позволяет несколько упростить исходные тексты и сделать их более понятными за счет уменьшения количества преобразований типов данных. Проиллюстрируем это на простом примере — сравним обращение к методу `Add3DFace` двумя способами:

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 0bbf0d44>
;;; Получили указатель на пространство модели.
(vlax-invoke-method obj 'Add3DFace (vlax-3D-point (getpoint))
    (vlax-3D-point (getpoint)) (vlax-3D-point (getpoint)))
```

```
;;; #<VLA-OBJECT IAcad3DFace 05860ca4>
;;; Создали новый объект 3DFace.
(vlax-invoke obj 'Add3DFace (getpoint) (getpoint) (getpoint))
;;; #<VLA-OBJECT IAcad3DFace 05854054>
;;; Создали еще один новый объект 3DFace другим способом.
```

Вызов `vlax-invoke` аналогичен вызову `vlax-invoke-method`. Отличием является то, что в качестве аргументов `vlax-invoke` фигурируют типы данных AutoLISP, а не ActiveX. Фактически эта функция производит автоматическое преобразование типов данных в нужный формат. Тем не менее использование этой функции нельзя рекомендовать ввиду неясности ее дальнейшей судьбы.

При работе с объектами AutoCAD целесообразнее применять именные функции. Это, во-первых, позволяет визуально контролировать правильность написания их имен, а во-вторых, эти функции имеют меньшее количество аргументов, что упрощает их интерпретацию. Именно такой способ мы преимущественно и используем при обращении к методам.

Следует учитывать, что привести примеры всех возможных вариантов применения методов совершенно невозможно и, скорее всего, не нужно. При выборе примеров для демонстрации, как правило, предпочтение отдавалось ситуациям, наиболее востребованным, по мнению авторов, в реальном программировании. Использование некоторых методов обуславливается соответствующим контекстом выполнения. Как правило, этот контекст заложен в код приведенных примеров без отдельного текстового описания, с расчетом на то, что читатель обладает необходимой квалификацией для того, чтобы прочитать и разобрать демонстрационный код.

В некоторых случаях описание функций, приведенное в книге, расходится с тем, что можно найти в официальной документации или материалах, основанных исключительно на ее изучении. Авторы приводят то, что наблюдали своими глазами, а не то, что где-то прочитали. Кроме того, часть методов официально не документирована, а часть в рамках Visual LISP выполнить на практике не удалось. Вероятными причинами могут быть ошибки реализации механизмов ActiveX в интерпретаторе LISP или то, что авторам не удалось создать необходимый контекст использования этих методов.

В рамках данной книги рассматриваются объекты только базовой версии AutoCAD. Некоторые вертикальные приложения, в особенности разработанные непосредственно фирмой Autodesk, имеют свои объекты ActiveX. Такие объекты, как правило, имеют свой набор методов и свойств, в том числе и таких, которые у стандартных объектов не встречаются. Ввиду отсутствия соответствующих именных функций, доступ к таким свойствам осуществляется по правилам доступа к свойствам объектов другого приложения.

В описании функций, реализующих методы ActiveX, мы используем следующую форму изложения. Сначала даются названия объектов, к которым применим этот

метод, потом приводится назначение, затем аргументы и возможные варианты возвращаемых значений. Могут фигурировать зарезервированные константы технологии ActiveX. Завершается описание каждого метода примером его применения. Все примеры, включая самые простые и очевидные, в ходе подготовки книги были выполнены авторами фактически и приведены только реально полученные результаты их выполнения.

Методы перечислены в алфавитном порядке. В квадратных скобках приведены необязательные аргументы. Термин "возврат" означает "возвращаемое значение". Примечания относятся к абзацам, непосредственно за которыми они расположены.

7.2.1. Метод *Activate*

Объект: **Document**.

Описание: активация документа.

Аргументы: нет. Возврат: `nil`.

Примечание

При переключении активного документа выполнение функции, содержащей обращение к методу `Activate`, приостанавливается до момента возвращения в тот документ, из которого она была вызвана.

```
(setq obj (vla-Item (vla-get-Documents (vlax-get-acad-object)) 1))
;;; #<VLA-OBJECT IAcadDocument 0c36ef90>
(vla-Activate obj)
```

7.2.2. Метод *Add*

Объекты: **Blocks**, **Dictionaries**, **DimStyles**, **Documents**, **Groups**, **Hyperlinks**, **KeyDescriptions**, **KeyValues**, **LabelTemplates**, **Layers**, **Layouts**, **Linetypes**, **LinkTemplates**, **PlotConfigurations**, **PopupMenus**, **Projects**, **Queries**, **RegisteredApplications**, **SelectionSets**, **TextStyles**, **Toolbars**, **UCSs**, **Viewports**, **Views**.

Описание: создает объект и добавляет его в соответствующее семейство.

Аргументы (различные случаи):

- **Blocks** (создается пустой блок)
  - `<точка вставки>` — координаты трехмерной точки в виде варианта с типом безопасного массива;
  - `<имя>` — строка с именем блока; для анонимного блока — строка типа `"*U"`.
- **UCSs**
  - `<нулевая точка>` — координаты трехмерной точки в виде варианта с типом безопасного массива;

- <положительное направление оси X> — координаты трехмерной точки в виде варианта с типом безопасного массива (не может совпадать с нулевой точкой);
- <положительное направление оси Y> — координаты трехмерной точки в виде варианта с типом безопасного массива (не может совпадать с нулевой точкой, оси должны быть перпендикулярны);
- <имя> — строка с именем ПСК.

♦ **Hyperlinks**

- <адрес> — строка с адресом ссылки;
- [<пояснение>] — строковый вариант;
- [<якорь>] — строковый вариант.

♦ **PlotConfigurations**

- <имя> — строка с именем конфигурации печати;
- [<тип>] — применимость:

:vlax-true — только к вкладке **Model** (Модель);

:vlax-false — ко всем вкладкам.

♦ **Dictionaries**, **DimStyles**, **Documents**, **Groups**, **Layers**, **Layouts**, **Linetypes**, **PopupMenus**, **Projects**, **RegisteredApplications**, **SelectionSets**, **TextStyles**, **Toolbars**, **Viewports**, **Views**

- <имя> — строка с именем элемента (словаря, размерного стиля, группы и т. п.).

♦ **KeyDescription**

- <имя> — строка;
- <тип данных> — формат, см. свойство Type;
- [<точность>] — целое число, актуально для числовых (точность данных), текстовых (длина строки символов) и бинарных (размер поля) типов данных;
- [<масштаб>] — целое число, актуально для числового типа данных.

♦ **KeyValue**

- <имя> — строка;
- <значение> — любой допустимый тип данных.

♦ **LabelTemplate**

- <источник>, <каталог>, <имя схемы>, <имя таблицы>, <имя> — строка;
- <отображаемые атрибуты> — форматированная строка вида #(<колонка 1>\P#(<колонка 2>).

- **LinkTemplate**
  - <источник>, <каталог>, <имя схемы>, <имя таблицы>, <имя> — строка;
  - <описание ключа> — VLA-объект.
- **Query**
  - <источник>, <каталог>, <имя схемы>, <имя таблицы>, <имя> — строка;
  - <запрос> — форматированная строка SQL-запроса.

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-Blocks actdoc))
;;; #<VLA-OBJECT IAcadBlocks 12515614>
(vla-Add obj (vlax-3D-point (getpoint)) "*U")
;;; #<VLA-OBJECT IAcadBlock2 12527fa4>
;;; Создан пустой безымянный блок.
(setq obj (vla-get-UserCoordinateSystems actdoc))
;;; #<VLA-OBJECT IAcadUCSs 12527cb4>
(vla-Add obj (vlax-3D-point (getpoint)) (vlax-3D-point (getpoint))
    (vlax-3D-point (getpoint)) "TestUCS")
;;; #<VLA-OBJECT IAcadUCS 0c33a564>
;;; Создана ПСК. Оси должны быть перпендикулярны.
(setq obj (vla-get-Hyperlinks (vlax-ename->vla-object (car (entsel)))))
;;; #<VLA-OBJECT IAcadHyperlinks 0c369c74>
(vla-Add obj "http://www.uniip.ru" (vlax-make-variant "TestHyperlink"
    vlax-vbString) (vlax-make-variant "anch1" vlax-vbString))
;;; #<VLA-OBJECT IAcadHyperlink 1252f434>
;;; Создана гиперссылка с пояснением и якорем.
(setq DBConn (vla-GetInterfaceObject (vlax-get-acad-object)
               "CAO.DbConnect.16"))
;;; #<VLA-OBJECT IDbConnect 05873a20>
(setq DBLinksT (vlax-invoke DBConn 'GetLinkTemplates))
;;; #<VLA-OBJECT ILinkTemplates 05873c40>
(setq DBLinkT
;;; #<VLA-OBJECT ILinkTemplate 058733f0>
(setq KeyDeskrs (vlax-get-property DBLinkT 'KeyDescriptions))
;;; #<VLA-OBJECT IKeyDescriptions 058737e0>
(setq DBLinkT (vla-Add DBLinksT "jet_dbsamples" "Demo" "Test" "Computer"
  "Link7" KeysDskrs))
;;; #<VLA-OBJECT ILinkTemplate 05898ef0>
;;; Создан шаблон связи. Будет использоваться ниже.
(setq DBLablsT (vlax-invoke DBConn 'GetLabelTemplates))
;;; #<VLA-OBJECT ILabelTemplates 0589cf40>
(vla-Add DBLablsT "jet_dbsamples" "Demo" "Test" "Computer" "Label3"
  "Изготовитель: #(Manufacturer)\\P{\\fVerdana;#(Room)}")
```

```
;;; #<VLA-OBJECT ILabelTemplate 05874e10>
;;; Создан шаблон ярлыка.
(setq DBQueries (vlax-invoke DBConn 'GetQueries))
;;; #<VLA-OBJECT IQueries 01150820>
(vla-Add DBQueries "jet_dbsamples" "Demo" "Test" "Inventory" "Query2"
  "SELECT * FROM Inventory\r\n\tWHERE Price > 150")
;;; #<VLA-OBJECT IQuery 0115b020>
;;; Создан запрос.
(setq KeyDeskrs (vlax-get-property DBLinkT 'KeyDescriptions))
;;; #<VLA-OBJECT IKeyDescriptions 0115b2e0>
(vla-Add KeyDeskrs "KDeskr1" 1 20 nil)
;;; #<VLA-OBJECT IKeyDescription 0115a420>
;;; Создано описание ключа.
(setq DBLinks (vlax-invoke DBConn 'GetLinks))
;;; #<VLA-OBJECT ILinks 0115d530>
(setq DBLink (vla-Item DBLinks 2))
;;; #<VLA-OBJECT ILink 0115fd80>
(setq KeyVals (vlax-get-property DBLink 'KeyValues))
;;; #<VLA-OBJECT IKeyValues 0115d7a0>
(vla-Add KeyVals "KVal1" 15)
;;; #<VLA-OBJECT IKeyValue 0115a350>
;;; Создано значение ключа.
```

7.2.3. Метод *Add3DFace*

Объекты: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание объекта `3DFace`.

Аргументы:

- `<точка 1>`, `<точка 2>`, `<точка 3>` и [`<точка 4>`] — координаты трехмерной точки в виде варианта с типом безопасного массива.

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 0bbf0d44>
(vla-Add3DFace obj (vlax-3D-point (getpoint)) (vlax-3D-point (getpoint))
    (vlax-3D-point (getpoint)) (vlax-3D-point (getpoint)))
;;; #<VLA-OBJECT IAcad3DFace 0bbf0f14>
```

7.2.4. Метод *Add3DMesh*

Объекты: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание объекта `3DMesh`.

Аргументы:

- <размерность M>, <размерность N> — целое число от 2 до 256;
- <массив вершин> — одномерный безопасный массив координат трехмерных точек.

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 0bbf0d44>
(vla-Add3DMesh obj 2 3 (vlax-safearray-fill
    (vlax-make-safearray vlax-vbDouble '(0 . 17))
    (append (getpoint) (getpoint) (getpoint)
            (getpoint) (getpoint) (getpoint))))
;;; #<VLA-OBJECT IAcadPolygonMesh 0bbf1c74>
```

7.2.5. Метод *Add3DPoly*

Объекты: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание объекта 3DPolyline.

Аргументы:

- <массив вершин> — одномерный безопасный массив координат трехмерных точек.

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 0bbf0d44>
(vla-Add3DPoly obj (vlax-safearray-fill (vlax-make-safearray
  vlax-vbDouble '(0 . 8)) (append (getpoint) (getpoint) (getpoint))))
;;; #<VLA-OBJECT IAcad3DPolyline 0bbf0864>
```

7.2.6. Метод *AddArc*

Объекты: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание объекта Arc.

Аргументы:

- <центр> — координаты трехмерной точки в виде варианта с типом безопасного массива;
- <радиус>, <начальный угол> и <конечный угол> — положительные вещественные числа.

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 0bbf0d44>
(vla-AddArc obj (vlax-3D-point (getpoint)) 150 (/ pi 4) (* pi 1.5))
;;; #<VLA-OBJECT IAcadArc 0bbf0224>
```

7.2.7. Метод *AddAttribute*

Объекты: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание объекта `Attribute`.

Аргументы:

- `<высота>` — положительное вещественное число;
- `<тип>` — битовая маска (значения могут складываться):

  `acAttributeModeNormal` (0) — обычный;

  `acAttributeModeInvisible` (1) — невидимый;

  `acAttributeModeConstant` (2) — постоянный;

  `acAttributeModeVerify` (4) — проверяемый;

  `acAttributeModePreset` (8) — предустановленный.
- `<сообщение>` — строка;
- `<точка вставки>` — координаты трехмерной точки в виде варианта с типом безопасного массива;
- `<тег>` — строка;
- `<значение по умолчанию>` — строка.

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 0bbf0d44>
(vla-AddAttribute obj 20 acAttributeModeNormal "Test"
    (vlax-3D-point (getpoint)) "MYATR" "Any string")
;;; #<VLA-OBJECT IAcadAttribute 0bbf1c74>
```

7.2.8. Метод *AddBox*

Объекты: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание ящика (объекта `3DSolid`).

Аргументы:

- `<точка вставки>` — координаты трехмерной точки в виде варианта с типом безопасного массива;
- `<длина>`, `<ширина>` и `<высота>` — положительные вещественные числа.

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 0bbf0d44>
(vla-AddBox obj (vlax-3D-point (getpoint)) 300 250 600)
;;; #<VLA-OBJECT IAcad3DSolid 0bbefd34>
```

7.2.9. Метод *AddCircle*

Объекты: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание объекта `Circle`.

Аргументы:

- `<центр>` — координаты трехмерной точки в виде варианта с типом безопасного массива;
- `<радиус>` — положительное вещественное число.

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 0bbf0d44>
(vla-AddCircle obj (vlax-3D-point (getpoint)) 420)
;;; #<VLA-OBJECT IAcadCircle 0bbef324>
```

7.2.10. Метод *AddCone*

Объекты: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание конуса (объекта `3DSolid`).

Аргументы:

- `<центр>` — координаты трехмерной точки в виде варианта с типом безопасного массива;
- `<радиус основания>` и `<высота>` — положительные вещественные числа.

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 0bbf0d44>
(vla-AddCone obj (vlax-3D-point (getpoint)) 210 450)
;;; #<VLA-OBJECT IAcad3DSolid 0bbee034>
```

7.2.11. Метод *AddCustomInfo*

Объект: **SummaryInfo**.

Описание: создание параметра пользовательских свойств рисунка.

Аргументы:

- <имя поля> и <значение> — строки.

Возврат: nil.

```
(setq obj (vla-get-SummaryInfo actdoc))
;;; #<VLA-OBJECT IAcadSummaryInfo 0bbf0df4>
(vla-AddCustomInfo obj "My Data" "Important string")
```

7.2.12. Метод *AddCustomObject*

Объекты: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание пользовательского объекта DBX- или ARX-приложения.

Аргументы:

- <имя класса> — строка.

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 0bbf0d44>
(vla-AddCustomObject obj "AecWall")
;;; #<VLA-OBJECT IAecWall 0bc03680>
;;; Создан объект ADT.
```

7.2.13. Метод *AddCylinder*

Объекты: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание цилиндра (объекта 3DSolid).

Аргументы:

- <центр> — координаты трехмерной точки в виде варианта с типом безопасного массива;
- <радиус> и <высота> — положительные вещественные числа.

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 0bbf0d44>
(vla-AddCylinder obj (vlax-3D-point (getpoint)) 160 400)
;;; #<VLA-OBJECT IAcad3DSolid 0b4ca7a4>
```

7.2.14. Метод *AddDim3PointAngular*

Объекты: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание объекта Dim3PointAngular.

Аргументы:

- <вершина>, <точка 1>, <точка 2> и <точка вставки размерного текста> — координаты трехмерных точек в виде варианта с типом безопасного массива.

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 0bbf0d44>
(vla-AddDim3PointAngular obj (vlax-3D-point (getpoint))
    (vlax-3D-point (getpoint)) (vlax-3D-point (getpoint))
    (vlax-3D-point (getpoint)))
;;; #<VLA-OBJECT IAcadDim3PointAngular 0b4d4bc4>
```

7.2.15. Метод *AddDimAligned*

Объекты: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание объекта DimAligned.

Аргументы:

- <точка 1>, <точка 2> и <точка вставки размерного текста> — координаты трехмерной точки в виде варианта с типом безопасного массива.

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 0bbf0d44>
(vla-AddDimAligned obj (vlax-3D-point (getpoint))
    (vlax-3D-point (getpoint)) (vlax-3D-point (getpoint)))
;;; #<VLA-OBJECT IAcadDimAligned 0bc0b084>
```

7.2.16. Метод *AddDimAngular*

Объекты: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание объекта DimAngular.

Аргументы:

- <вершина>, <точка 1>, <точка 2> и <точка вставки размерного текста> — координаты трехмерных точек в виде варианта с типом безопасного массива.

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 0bbf0d44>
(vla-AddDimAngular obj (vlax-3D-point (getpoint))
    (vlax-3D-point (getpoint)) (vlax-3D-point (getpoint))
    (vlax-3D-point (getpoint)))
;;; #<VLA-OBJECT IAcadDimAngular 0bc167c4>
```

7.2.17. Метод *AddDimArc*

Объекты: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание объекта DimArcLength.

Аргументы:

- <центр>, <точка 1>, <точка 2>, <точка положения размерной дуги> — координаты трехмерных точек в виде вариантов с типом безопасного массива.

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace3 0b0f1984>
(vla-AddDimArc obj (vlax-3D-point (getpoint)) (vlax-3D-point (getpoint))
    (vlax-3D-point (getpoint)) (vlax-3D-point (getpoint)))
;;; #<VLA-OBJECT IAcadDimArcLength 0b0f17d4>
```

7.2.18. Метод *AddDimDiametric*

Объекты: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание объекта DimDiametric.

Аргументы:

- <точка 1> и <точка 2> — координаты трехмерной точки в виде варианта с типом безопасного массива;
- <длина выноски> — положительное вещественное число.

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 0bbf0d44>
(vla-AddDimDiametric obj (vlax-3D-point (getpoint))
    (vlax-3D-point (getpoint)) 50)
;;; #<VLA-OBJECT IAcadDimDiametric 05819cd4>
```

7.2.19. Метод *AddDimOrdinate*

Объекты: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание объекта DimOrdinate.

Аргументы:

- <базовая точка> и <конец выноски> — координаты трехмерных точек в виде варианта с типом безопасного массива;

- <координаты по оси>:

  :vlax-true — *X*;

  :vlax-false — *Y*.

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 0bbf0d44>
(vla-AddDimOrdinate obj (vlax-3D-point (getpoint))
    (vlax-3D-point (getpoint)) :vlax-false)
;;; #<VLA-OBJECT IAcadDimOrdinate 0bc15c94>
```

7.2.20. Метод *AddDimRadial*

Объекты: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание объекта DimRadial.

Аргументы:

- <центр> и <точка размера> — координаты трехмерных точек в виде варианта с типом безопасного массива;
- длина выноски — положительное вещественное число.

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 0bbf0d44>
(vla-AddDimRadial obj (vlax-3D-point (getpoint))
    (vlax-3D-point (getpoint)) 50)
;;; #<VLA-OBJECT IAcadDimRadial 0bc1d304>
```

7.2.21. Метод *AddDimRadialLarge*

Объекты: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание объекта DimRadialLarge.

Аргументы:

- <центр>, <точка начала выноски>, <точка мнимого центра>, <точка положения зигзага> — координаты трехмерных точек в виде вариантов с типом безопасного массива;
- <угол зигзага> — вещественное число.

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace3 0b0f1984>
```

```
(vla-AddDimRadialLarge obj (vlax-3D-point (getpoint))
 (vlax-3D-point (getpoint)) (vlax-3D-point (getpoint))
 (vlax-3D-point (getpoint)) (/ pi 4))
;;; #<VLA-OBJECT IAcadDimRadialLarge 0b0fa3d4>
```

7.2.22. Метод *AddDimRotated*

Объекты: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание объекта `DimRotated`.

Аргументы:

- `<точка 1>`, `<точка 2>` и `<точка вставки размерного текста>` — координаты трехмерных точек в виде варианта с типом безопасного массива;
- `<угол поворота>` — положительное вещественное число.

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 0bbf0d44>
(vla-AddDimRotated obj (vlax-3D-point (getpoint))
    (vlax-3D-point (getpoint)) (vlax-3D-point (getpoint)) (/ pi 6))
;;; #<VLA-OBJECT IAcadDimRotated 0bc19de4>
```

7.2.23. Метод *AddEllipse*

Объекты: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание объекта `Ellipse`.

Аргументы:

- `<центр>` — координаты трехмерной точки в виде варианта с типом безопасного массива;
- `<главная ось>` — координаты смещения конца главной оси относительно центра в виде варианта с типом безопасного массива;
- `<соотношение осей>` — вещественное число больше 0 до 1.0 включительно.

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 0c299e04>
(vla-AddEllipse obj (vlax-3D-point (getpoint))
    (vlax-3D-point '(100 75 0)) 0.7)
;;; #<VLA-OBJECT IAcadEllipse 0c2d2a54>
```

7.2.24. Метод *AddEllipticalCone*

Объекты: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание эллиптического конуса (объекта `3DSolid`).

Аргументы:

- `<центр>` — координаты трехмерной точки в виде варианта с типом безопасного массива;
- `<радиус по X>`, `<радиус по Y>` и `<высота>` — положительные вещественные числа.

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 0bbf0d44>
(vla-AddEllipticalCone obj (vlax-3D-point (getpoint)) 120 160 300)
;;; #<VLA-OBJECT IAcad3DSolid 0c2d3c54>
```

7.2.25. Метод *AddEllipticalCylinder*

Объекты: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание эллиптического цилиндра (объекта `3DSolid`).

Аргументы:

- `<центр>` — координаты трехмерной точки в виде варианта с типом безопасного массива;
- `<радиус по X>`, `<радиус по Y>` и `<высота>` — положительные вещественные числа.

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 0bbf0d44>
(vla-AddEllipticalCylinder obj (vlax-3D-point (getpoint)) 80 65 20)
;;; #<VLA-OBJECT IAcad3DSolid 0c2d2174>
```

7.2.26. Метод *AddExtrudedSolid*

Объекты: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание объекта `3DSolid` способом выдавливания.

Аргументы:

- `<профиль>` — VLA-объект профиля, только `Region`;
- `<высота>` — ненулевое вещественное число в диапазоне ±100000, знак соответствует направлению оси *Z*;

- <наклон граней> — вещественное число в пределах ±π/2.00000127324035514859 1, в радианах. Геометрия создаваемого объекта должна быть без самопересечений, ограничение по размеру объекта по осям *X* и *Y* ~50000.

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 05ba1dd4>
(setq temp (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadRegion 05b54da4>
(vla-AddExtrudedSolid obj temp 80 (/ pi -6))
;;; #<VLA-OBJECT IAcad3DSolid 09e6a574>
```

7.2.27. Метод *AddExtrudedSolidAlongPath*

Объекты: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание объекта 3DSolid способом выдавливания по траектории.

Аргументы:

- <профиль> — VLA-объект (профиль), только Region;
- <траектория> — VLA-объект (путь), могут быть LightweightPolyline, Polyline, Circle, Ellipse, Spline или Arc.

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 05ba1dd4>
(setq temp (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadRegion 05b54da4>
(setq path (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadLWPolyline 0c222594>
(vla-AddExtrudedSolidAlongPath obj temp path)
;;; #<VLA-OBJECT IAcad3DSolid 0c225ef4>
```

7.2.28. Метод *AddFitPoint*

Объект: **Spline**.

Описание: добавление опорной точки к сплайну по номеру.

Аргументы:

- <индекс> — положительное целое число, номер добавляемой точки;
- <точка> — координаты трехмерной точки в виде варианта с типом безопасного массива.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadSpline 0c046204>
(vla-AddFitPoint obj 3 (vlax-3D-point (getpoint)))
```

7.2.29. Метод *AddHatch*

Объекты: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание объекта `Hatch`.

Аргументы:

- `<тип>` — целое число или константа:

  `acHatchPatternTypeUserDefined` (0) — пользовательская;

  `acHatchPatternTypePreDefined` (1) — предопределенная (`acad.pat`);

  `acHatchPatternTypeCustomDefined` (2) — определенная в ином файле.
- `<имя образца>` — строка;
- `<признак ассоциативности>`:

  `:vlax-true` — да;

  `:vlax-false` — нет.

Примечание

Свойство ассоциативности для штриховки созданной, как "ассоциативная", может быть ликвидировано, но не может быть восстановлено. Возможности изменения свойств неассоциативной штриховки средствами ActiveX минимальны.

- `[<тип заполнения>]` — целое число или константа:

  `AcHatchObject` (0) — обычная штриховка или SOLID;

  `AcGradientObject` (1) — градиентная заливка.

Возврат: созданный VLA-объект.

Примечание

Применение данного метода приводит к созданию недоопределенного объекта — штриховки без границ. Для завершения определения штриховки необходимо применить к ней метод `AppendOuterLoop` (рассматривается отдельно), в противном случае AutoCAD остается в непредсказуемом состоянии. Во избежание неприятностей, в данном примере штриховка удаляется сразу после создания.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 0c031974>
(setq temp (vla-AddHatch obj acHatchPatternTypePreDefined
    "ansi32" :vlax-true AcHatchObject))
;;; #<VLA-OBJECT IAcadHatch 0c05f244>
(vla-Delete temp)
```

7.2.30. Метод *AddItems*

Объект: **SelectionSet**.

Описание: добавление графических объектов в набор.

Аргументы:

- <объекты> — вариант (безопасный массив), содержащий объекты.

Возврат: nil.

```
(setq sel (vla-get-ActiveSelectionSet actdoc))
;;; #<VLA-OBJECT IAcadSelectionSet 0c548d94>
;;; получили безымянный набор
(vla-Clear sel)
;;; очистили его, на всякий случай (только для этого примера)
(setq temp (list (vlax-ename->vla-object (car (entsel)))))
;;; (#<VLA-OBJECT IAcadEllipse 0c548d34>)
;;; создали список из объектов, которые хотим добавить
(setq temp (vlax-make-variant (vlax-safearray-fill
    (vlax-make-safearray vlax-vbObject '(0 . 0)) temp)))
;;; #<variant 8201 ...>
;;; создали массив из списка
(vla-AddItems sel temp)
```

7.2.31. Метод *AddLeader*

Объекты: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание объекта Leader.

Аргументы:

- <точки> — вариант (плоский безопасный массив), содержащий координаты трехмерных точек (не менее двух);
- <аннотация> — объект типа Tolerance, MText или BlockRef;

Примечание

Средствами Visual LISP создать выноску без аннотации не представляется возможным ввиду отсутствия объекта типа null. Данная задача может быть реализована при помощи AutoLISP.

- <тип> — целое число или константа:

  acLineNoArrow (0) — ломаная, без стрелки;

  acSplineNoArrow (1) — сплайновая, без стрелки;

  acLineWithArrow (2) — ломаная, со стрелкой;

  acSplineWithArrow (3) — сплайновая, со стрелкой.

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 0c54bcc4>
(setq temp (append (getpoint) (getpoint) (getpoint)))
;;; (1512.89 -544.702 0.0 1591.5 -221.574 0.0 1389.82 16.8567 0.0)
;;; создали плоский список из трех точек
(setq temp (vlax-make-variant (vlax-safearray-fill
    (vlax-make-safearray vlax-vbDouble '(0 . 8)) temp)))
;;; #<variant 8197 ...>
;;; создали массив координат
(setq annot (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadBlockReference 0c5499c4>
;;; выбрали объект для аннотации
(vla-AddLeader obj temp annot acSplineWithArrow)
;;; #<VLA-OBJECT IAcadLeader 0c54b934>
```

7.2.32. Метод *AddLightWeightPolyline*

Объекты: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание объекта `LightWeightPolyline`.

Аргументы:

- `<точки>` — вариант (плоский безопасный массив), содержащий координаты двумерных точек (не менее двух).

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 0c54bcc4>
(setq temp (list (getpoint) (getpoint) (getpoint)))
;;; ((528.095 -696.687 0.0) (267.423 -295.895 0.0) (531.58 378.061 0.0))
;;; создали список из трех трехмерных точек
(setq temp (apply 'append (mapcar 'list (mapcar 'car temp)
    (mapcar 'cadr temp))))
;;; (528.095 -696.687 267.423 -295.895 531.58 378.061)
;;; преобразовали список точек к двумерному плоскому
(setq temp (vlax-make-variant (vlax-safearray-fill
    (vlax-make-safearray vlax-vbDouble '(0 . 5)) temp)))
;;; #<variant 8197 ...>
;;; создали массив координат
(vla-AddLightWeightPolyline obj temp)
;;; #<VLA-OBJECT IAcadLWPolyline 0c544084>
```

7.2.33. Метод *AddLine*

Объекты: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание объекта `Line`.

Аргументы:

- <точка 1> и <точка 2> — координаты трехмерных точек в виде варианта с типом безопасного массива.

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 0bbf0d44>
(vla-AddLine obj (vlax-3D-point (getpoint)) (vlax-3D-point (getpoint)))
;;; #<VLA-OBJECT IAcadLine 0c547144>
```

7.2.34. Метод *AddMenuItem*

Объект: **PopupMenu**.

Описание: добавление пункта падающего или контекстного меню.

Аргументы:

- <индекс> — целое число или строка;

Примечание

Если аргумент — целое число, то оно интерпретируется, как позиция нового элемента в падающем меню, нумерация начинается с нуля. Значение не должно быть отрицательным. Если значение превышает количество имеющихся элементов, то новый элемент добавляется последним. Если передана строка, то новый элемент добавляется перед тем, заголовок которого указан, как индекс или в конец меню, если элемента с таким заголовком нет.

- <заголовок> — строка;
- <макрос> — строка (выполняемый макрос), может содержать выражения языка DIESEL.

Возврат: созданный VLA-объект.

```
(setq obj (vla-Item (vla-get-Menus (vla-Item (vla-get-MenuGroups
    (vlax-get-acad-object)) "acad")) 9))
;;; #<VLA-OBJECT IAcadPopupMenu 010ae678>
(setq temp (vla-AddMenuItem obj 3 "Test" "\003\003test "))
;;; #<VLA-OBJECT IAcadPopupMenuItem 7ef55fe8>
(vla-Delete temp)
;;; Удалили пункт, чтобы не замусоривать меню пробными элементами.
```

7.2.35. Метод *AddMInsertBlock*

Объекты: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание объекта MInsertBlock.

Аргументы:

- <точка вставки> — координаты трехмерной точки в виде варианта с типом безопасного массива;
- <имя> — строка;
- <масштаб X>, <масштаб Y>, <масштаб Z> — масштабные коэффициенты по осям, вещественные числа;
- <угол поворота> — вещественное число, угол в радианах;
- <строки>, <колонки> — положительные целые числа;
- <шаг строк>, <шаг колонок> — вещественные числа;
- [<пароль>] — строка.

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 0bbf0d44>
(vla-AddMInsertBlock obj (vlax-3D-point (getpoint)) "V1" 1.2 0.9 1.2
    (/ pi 8) 2 3 100 120)
;;; #<VLA-OBJECT IAcadMInsertBlock 0c0eb5d4>
```

7.2.36. Метод *AddMLine*

Объекты: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание объекта MLine.

Аргументы:

- <массив вершин> — одномерный безопасный массив координат трехмерных точек.

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 0bbf0d44>
(vla-AddMLine obj (vlax-safearray-fill
    (vlax-make-safearray vlax-vbDouble '(0 . 8))
    (append (getpoint) (getpoint) (getpoint))))
;;; #<VLA-OBJECT IAcadMLine 0c0efe74>
```

7.2.37. Метод *AddMText*

Объекты: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание объекта MText.

Аргументы:

- <точка вставки> — координаты трехмерной точки в виде варианта с типом безопасного массива;
- <ширина> — вещественное число;
- <текст> — строка.

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 0bbf0d44>
(vla-AddMText obj (vlax-3D-point (getpoint)) 280 "Пробный текст\nEND")
;;; #<VLA-OBJECT IAcadMText2 0c0ef174>
```

7.2.38. Метод *AddObject*

Объект: **Dictionary**.

Описание: создание объекта в словаре.

Аргументы:

- <ключ> — строка, идентификатор объекта;
- <имя класса объекта> — строка.

Возврат: VLA-объект.

Примечание

Имя класса объекта определяется родительским приложением. Основной класс — AcDbDictionary.

```
(setq obj (vla-Add (vla-get-Dictionaries actdoc) "MyTestDict"))
;;; #<VLA-OBJECT IAcadDictionary 0c593b84>
(vla-AddObject obj "TestKey" "AcDbTableStyle")
;;; #<VLA-OBJECT IAcadTableStyle 0c58b2a4>
```

7.2.39. Метод *AddPoint*

Объекты: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание объекта Point.

Аргументы:

- <точка вставки> — координаты трехмерной точки в виде варианта с типом безопасного массива.

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 0c58cd44>
(vla-AddPoint obj (vlax-3D-point (getpoint)))
;;; #<VLA-OBJECT IAcadPoint 0c58b054>
```

7.2.40. Метод *AddPolyfaceMesh*

Объекты: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание объекта `PolyfaceMesh`.

Аргументы:

- `<массив вершин>` — одномерный массив координат трехмерных точек в виде безопасного массива;
- `<массив граней>` — одномерный массив целых чисел, индексов вершин, образующих грани (количество кратно 4, где каждые 4 числа образуют грань).

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 0c58cd44>
(vla-AddPolyfaceMesh obj (vlax-safearray-fill
    (vlax-make-safearray vlax-vbDouble '(0 . 17))
    '(0 140 0 100 140 80 200 140 20 0 10 0 100 10 80 200 10 20))
    (vlax-safearray-fill (vlax-make-safearray vlax-vbInteger '(0 . 19))
    '(1 2 5 4 2 3 6 5 1 2 3 3 4 5 6 6 1 3 6 4))) ;_ соответствует призме
;;; #<VLA-OBJECT IAcadPolyfaceMesh 0c5a9924>
```

7.2.41. Метод *AddPolyline*

Объекты: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание объекта `Polyline`.

Аргументы:

- `<массив вершин>` — одномерный массив координат трехмерных точек в виде безопасного массива.

Примечание

Координата *Z* данным методом игнорируется. Метод сохранен только для обратной совместимости.

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 0bbf0d44>
```

```
(vla-AddPolyline obj (vlax-safearray-fill
    (vlax-make-safearray vlax-vbDouble '(0 . 8))
    (append (getpoint) (getpoint) (getpoint))))
;;; #<VLA-OBJECT IAcadPolyline 0c5bffa4>
```

7.2.42. Метод *AddPViewport*

Объект: **PaperSpace**.

Описание: создание объекта PViewport.

Аргументы:

- <центр> — координаты трехмерной точки в виде варианта с типом безопасного массива;
- <ширина> и <высота> — вещественные числа.

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-PaperSpace actdoc))
;;; #<VLA-OBJECT IAcadPaperSpace2 0c5dff54>
(setq temp (vla-AddPViewport obj (vlax-3D-point (getpoint)) 150 100))
;;; #<VLA-OBJECT IAcadPViewport2 0c6069d4>
(vla-put-ViewportOn temp :vlax-true) ;_ включаем видовой экран в листе
```

7.2.43. Метод *AddRaster*

Объект: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание объекта Raster.

Аргументы:

- <файл> — строка, имя растрового файла;
- <точка вставки> — координаты трехмерной точки в виде варианта с типом безопасного массива, левый нижний угол;
- <масштаб> — положительное вещественное число;
- <угол поворота> — вещественное число, угол в радианах.

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 0bc73234>
(vla-AddRaster obj "C:/Doc ... pict.bmp" (vlax-3D-point (getpoint))
    150 (/ pi 8))
;;; #<VLA-OBJECT IAcadRasterImage 0c073e94>
```

7.2.44. Метод *AddRay*

Объект: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание объекта Ray.

Аргументы:

- <точка начала> и <точка направления> — координаты трехмерных точек в виде варианта с типом безопасного массива.

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 0bc73234>
(vla-AddRay obj (vlax-3D-point (getpoint)) (vlax-3D-point (getpoint)))
;;; #<VLA-OBJECT IAcadRay 0c073f84>
```

7.2.45. Метод *AddRegion*

Объект: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание объекта Region.

Аргументы:

- <граничные объекты> — массив объектов, образующих замкнутую область в виде варианта с типом безопасного массива; допустимы следующие объекты: Line, Arc, Circle, Elliptical Arc, LightweightPolyline, Spline.

Примечание

Примитивы должны образовывать один или несколько замкнутых контуров.

Возврат: массив созданных VLA-объектов.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 0bc73234>
(setq temp (mapcar '(lambda (a) (vlax-ename->vla-object (car a)))
    (list (entsel) (entsel) (entsel))))
;;; (#<VLA-OBJECT IAcadLWPolyline 0c0a9ee4>
;;; #<VLA-OBJECT IAcadArc 0c0aef34> #<VLA-OBJECT IAcadCircle 0c0aee04>)
(setq temp (vlax-safearray-fill (vlax-make-safearray vlax-vbObject
    '(0 . 2)) temp))
;;; #<safearray...>
(setq res (vla-AddRegion obj temp))
;;; #<variant 8201 ...>
(vlax-safearray->list (vlax-variant-value res))
;;; (#<VLA-OBJECT IAcadRegion 062de3f4>
;;; #<VLA-OBJECT IAcadRegion 062d9614>)
```

7.2.46. Метод *AddRevolvedSolid*

Объекты: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание объекта 3DSolid способом вращения.

Аргументы:

- <профиль> — VLA-объект (профиль), только Region;
- <положение оси> — координаты трехмерной точки, через которую проходит ось, в виде варианта с типом безопасного массива;
- <направление оси> — координаты трехмерной точки, указывающей направление оси от начала координат, в виде варианта с типом безопасного массива;
- <сектор заполнения> — угол в радианах, вещественное число в пределах ±2π, геометрия создаваемого объекта должна быть без самопересечений.

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 0bf18e34>
(setq temp (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadRegion 0bf38434>
(vla-AddRevolvedSolid obj temp (vlax-3D-point (getpoint))
    (vlax-3D-point '(0 1 0)) (* pi 2))
;;; #<VLA-OBJECT IAcad3DSolid 0bf409b4>
```

7.2.47. Метод *AddSeparator*

Объекты: **PopupMenu**, **Toolbar**.

Описание: создание объекта-разделителя PopupMenuItem или ToolbarItem.

Аргументы:

- <индекс> — целое число или строка.

Примечание

Аналогичен методу AddMenuItem.

Возврат: созданный VLA-объект.

```
(setq obj (vla-Item (vla-get-Menus (vla-Item (vla-get-MenuGroups
    (vlax-get-acad-object)) "ACAD")) 9))
;;; #<VLA-OBJECT IAcadPopupMenu 010ae6a8>
(setq temp (vla-AddSeparator obj 8))
;;; #<VLA-OBJECT IAcadPopupMenuItem 7ef55fe8>
(vla-Delete temp)
;;; Удалили разделитель, чтобы не замусоривать меню пробными элементами.
```

7.2.48. Метод *AddShape*

Объекты: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание объекта Shape.

Аргументы:

- <имя> — строка;
- <точка вставки> — координаты трехмерной точки в виде варианта с типом безопасного массива;
- <масштаб> — положительное вещественное число;
- <угол поворота> — вещественное число, угол в радианах.

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 103bc684>
(vla-AddShape obj "CIRC1" (vlax-3D-point (getpoint)) 5 (/ pi 8))
;;; #<VLA-OBJECT IAcadShape 103bb094>
```

7.2.49. Метод *AddSolid*

Объекты: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание объекта Solid.

Аргументы:

- <точка1>, <точка2>, <точка3>, <точка4> — координаты трехмерных точек в виде вариантов с типом безопасного массива.

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 103bc684>
(vla-AddSolid obj (vlax-3D-point (getpoint)) (vlax-3D-point (getpoint))
    (vlax-3D-point (getpoint)) (vlax-3D-point (getpoint)))
;;; #<VLA-OBJECT IAcadSolid 103bcb74>
```

7.2.50. Метод *AddSphere*

Объекты: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание объекта-шара (3DSolid).

Аргументы:

- <центр> — координаты трехмерной точки в виде варианта с типом безопасного массива;
- <радиус> — положительное вещественное число.

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 103bc684>
(vla-AddSphere obj (vlax-3D-point (getpoint)) 180)
;;; #<VLA-OBJECT IAcad3DSolid 103bb824>
```

7.2.51. Метод *AddSpline*

Объекты: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание объекта Spline.

Аргументы:

- <массив вершин> — одномерный массив координат трехмерных точек в виде безопасного массива;
- <начальная касательная>, <конечная касательная> — координаты трехмерных точек в виде вариантов с типом безопасного массива.

Примечание

Направления касательных определяются от начала координат. Для начальной касательной значение является обратным.

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 103bc684>
(vla-AddSpline obj (vlax-safearray-fill
    (vlax-make-safearray vlax-vbDouble '(0 . 8))
    (append (getpoint) (getpoint) (getpoint)))
    (vlax-3D-point (getpoint)) (vlax-3D-point (getpoint)))
;;; #<VLA-OBJECT IAcadSpline 103d1484>
```

7.2.52. Метод *AddSubMenu*

Объект: **PopupMenu**.

Описание: динамическое добавление подменю.

Аргументы:

- <индекс> — целое число или строка;

Примечание

Аналогичен методу AddMenuItem.

- <заголовок> — строка.

Возврат: созданный VLA-объект.

```
(setq obj (vla-Item (vla-get-Menus (vla-Item (vla-get-MenuGroups
    (vlax-get-acad-object)) "ACAD")) "&Modify"))
;;; #<VLA-OBJECT IAcadPopupMenu 010ae8a8>
(setq submenu (vla-AddSubMenu obj 3 "Test"))
;;; #<VLA-OBJECT IAcadPopupMenu 0c0f1f28>
```

Примечание

AutoCAD 2006 не позволяет корректно отобразить пустое подменю. В связи с этим после выполнения предыдущей строки примера должны обязательно следовать строки с добавляемыми пунктами этого подменю и установкой признака конца подменю в последнем пункте.

```
(setq temp (vla-AddMenuItem submenu 0 "My command" "MYMAC")) ;_ для 2006
;;; #<VLA-OBJECT IAcadPopupMenuItem 7ef67aa8>
(vla-put-EndSubMenuLevel temp 1) ; признак конца подменю
(vla-Delete (vla-Item obj 3))
;;; Удалили подменю, чтобы не замусоривать меню пробными элементами.
```

7.2.53. Метод *AddTable*

2005

Объекты: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание объекта `Table`.

Аргументы:

- `<точка вставки>` — координаты трехмерной точки в виде варианта с типом безопасного массива;
- `<количество строк>`, `<количество колонок>` — положительные целые числа;
- `<размер строк>`, `<размер колонок>` — положительные вещественные числа.

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 04e22ed4>
(vla-AddTable obj (vlax-3D-point (getpoint)) 5 4 30 120)
;;; #<VLA-OBJECT IAcadTable 04e2a1a4>
```

7.2.54. Метод *AddText*

Объекты: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание объекта `Text`.

Аргументы:

- `<значение>` — строка;

- <точка вставки> — координаты трехмерной точки в виде варианта с типом безопасного массива;
- <высота> — положительное вещественное число.

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 04e22ed4>
(vla-AddText obj "Test" (vlax-3D-point (getpoint)) 25)
;;; #<VLA-OBJECT IAcadText2 09cb40a4>
```

7.2.55. Метод *AddTolerance*

Объекты: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание объекта Tolerance.

Аргументы:

- <значение> — строка;
- <точка вставки>, <направление> — координаты трехмерной точки в виде варианта с типом безопасного массива.

Примечание

Направление определяется от начала координат.

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 04e22ed4>
(vla-AddTolerance obj "Test" (vlax-3D-point (getpoint))
    (vlax-3D-point (getpoint)))
;;; #<VLA-OBJECT IAcadTolerance 09cb5b04>
```

7.2.56. Метод *AddToolbarButton*

Объект: **Toolbar**.

Описание: добавление кнопки в панель инструментов.

Аргументы:

- <индекс> — целое число или строка;

Примечание

Аналогичен методу AddMenuItem.

- <заголовок> — строка;

- <подсказка> — строка, отображаемая в зоне статуса;
- <макрос> — строка, выполняемый макрос, может содержать выражения языка DIESEL.
- [<признак групповой кнопки>]:

  :vlax-true — групповая (подменю);

  :vlax-false — обычная.

Возврат: созданный VLA-объект.

```
(setq obj (vla-Item (vla-get-Toolbars (vla-Item (vla-get-MenuGroups
    (vlax-get-acad-object)) "ACAD")) "Modify"))
;;; #<VLA-OBJECT IAcadToolbar 010cf5fc>
(setq temp (vla-AddToolbarButton obj 3 "Test" "Help" "\003\003test "))
;;; #<VLA-OBJECT IAcadToolbarItem 0c1bf33c>
(vla-Delete temp)
;;; Удалили кнопку, чтобы не замусоривать меню пробными элементами.
```

7.2.57. Метод *AddTorus*

Объекты: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание тора (объекта 3DSolid).

Аргументы:

- <центр> — координаты трехмерной точки в виде варианта с типом безопасного массива;
- <радиус тора>, <радиус трубы> — положительные вещественные числа.

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 0c1dcf54>
(vla-AddTorus obj (vlax-3D-point (getpoint)) 180 20)
;;; #<VLA-OBJECT IAcad3DSolid 0c1db2b4>
```

7.2.58. Метод *AddTrace*

Объекты: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание объекта Trace.

Аргументы:

- <массив вершин> — одномерный безопасный массив координат четырех трехмерных точек.

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 0c1dcf54>
(vla-AddTrace obj (vlax-safearray-fill
    (vlax-make-safearray vlax-vbDouble '(0 . 11))
    (append (getpoint) (getpoint) (getpoint) (getpoint))))
;;; #<VLA-OBJECT IAcadTrace 0c1f6b44>
```

7.2.59. Метод *AddVertex*

Объект: **LightweightPolyline**.

Описание: добавление вершины.

Аргументы:

- <индекс> — номер добавляемой вершины, считая с нуля;
- <точка> — координаты двумерной точки в виде варианта с типом безопасного массива.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadLWPolyline 0c1c2aa4>
(vla-AddVertex obj 2 (vlax-safearray-fill
    (vlax-make-safearray vlax-vbDouble '(0 . 1))
    (reverse (cdr (reverse (getpoint))))))
```

7.2.60. Метод *AddWedge*

Объекты: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание клина (объекта 3DSolid).

Аргументы:

- <центр> — координаты трехмерной точки в виде варианта с типом безопасного массива;
- <длина>, <ширина>, <высота> — положительные вещественные числа.

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 0c1f3a44>
(vla-AddWedge obj (vlax-3D-point (getpoint)) 110 60 45)
;;; #<VLA-OBJECT IAcad3DSolid 0c1f6c14>
```

7.2.61. Метод *AddXline*

Объекты: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание объекта XLine.

Аргументы:

- <точка 1> и <точка 2> — координаты трехмерных точек в виде варианта с типом безопасного массива.

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 0c1f3a44>
(vla-AddXline obj (vlax-3D-point (getpoint)) (vlax-3D-point (getpoint)))
;;; #<VLA-OBJECT IAcadXline 0c1f40e4>
```

7.2.62. Метод *AddXRecord*

Объект: **Dictionary**.

Описание: добавление X-записи в словарь.

Аргументы:

- <ключ> — строка, идентификатор объекта.

Возврат: созданный VLA-объект.

```
(setq obj (vla-Add (vla-get-Dictionaries actdoc) "MyTestDict"))
;;; #<VLA-OBJECT IAcadDictionary 0c1f75a4>
(vla-AddXRecord obj "TestKey")
;;; #<VLA-OBJECT IAcadXRecord 0c1f7354>
```

7.2.63. Метод *AngleFromXAxis*

Объект: **Utility**.

Описание: получение угла от оси X в радианах.

Аргументы:

- <точка 1> и <точка 2> — координаты трехмерных точек в виде варианта с типом безопасного массива.

Возврат: положительное вещественное число.

```
(setq obj (vla-get-Utility actdoc))
;;; #<VLA-OBJECT IAcadUtility 0c11bc74>
(vla-AngleFromXAxis obj (vlax-3D-point (getpoint))
    (vlax-3D-point (getpoint)))
;;; 0.741027
```

7.2.64. Метод *AngleToReal*

Объект: **Utility**.

Описание: преобразование строкового представления угла в вещественное.

Аргументы:

- <угол> — строковое представление угла;
- <единицы> — входной формат представления угла:

  acDegrees (0) — десятичные градусы;

  acDegreeMinuteSeconds (1) — градусы, минуты, секунды;

  acGrads (2) — грады;

  acRadians (3) — радианы.

Возврат: положительное вещественное число, угол в радианах.

```
(setq obj (vla-get-Utility actdoc))
;;; #<VLA-OBJECT IAcadUtility 0c11bc74>
(vla-AngleToReal obj "123.46" acDegrees)
;;; 2.15478
```

7.2.65. Метод *AngleToString*

Объект: **Utility**.

Описание: преобразование вещественного представления угла в строковое.

Аргументы:

- <угол> — вещественное представление угла в радианах;
- <единицы> — выходной формат представления угла (см. метод AngleToReal);
- <точность> — целое число от 0 до 8, количество знаков после запятой.

Возврат: строка.

```
(setq obj (vla-get-Utility actdoc))
;;; #<VLA-OBJECT IAcadUtility 0c11bc74>
(vla-AngleToString obj 2.46 acDegreeMinuteSeconds 3)
;;; "140d56'51\""
```

7.2.66. Метод *AppendInnerLoop*

Объект: **Hatch**.

Описание: добавляет островок внутреннего контура к штриховке.

Примечание

При создании новой штриховки может быть вызван только после задания внешней границы штриховки, см. метод AppendOuterLoop.

Аргументы:

- <граница> — массив объектов, образующих один замкнутый островок. Допустимы следующие объекты: Line, Polyline, Circle, Ellipse, Spline, Region.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadHatch 0c13fac4>
(setq temp (mapcar '(lambda (a) (vlax-ename->vla-object (car a)))
    (list (entsel) (entsel) (entsel))))
;;; (#<VLA-OBJECT IAcadLine 0c13fb94> #<VLA-OBJECT IAcadLine 0c13f364>
;;; #<VLA-OBJECT IAcadLine 0c13f254>)
(setq temp (vlax-safearray-fill (vlax-make-safearray vlax-vbObject
    '(0 . 2)) temp))
;;; #<safearray...>
(vla-AppendInnerLoop obj temp)
```

7.2.67. Метод *AppendItems*

Объект: **Group**.

Описание: добавляет объекты в группу.

Аргументы:

- <объекты> — массив добавляемых объектов.

Возврат: nil.

```
(setq obj (vla-Item (vla-get-Groups actdoc) "Test"))
;;; #<VLA-OBJECT IAcadGroup 0c32c8c4>
(setq temp (mapcar '(lambda (a) (vlax-ename->vla-object (car a)))
    (list (entsel) (entsel) (entsel))))
;;; (#<VLA-OBJECT IAcadLWPolyline 0c32db54>
;;;  #<VLA-OBJECT IAcadLine 0c32da84> #<VLA-OBJECT IAcadCircle 0c32da24>)
(setq temp (vlax-safearray-fill (vlax-make-safearray vlax-vbObject
    '(0 . 2)) temp))
;;; #<safearray...>
(vla-AppendItems obj temp)
```

7.2.68. Метод *AppendOuterLoop*

Объект: **Hatch**.

Описание: добавляет островок внешнего контура к штриховке.

Аргументы:

- <граница> — массив объектов, образующих один замкнутый контур. Допустимы следующие объекты: Line, Polyline, Circle, Ellipse, Spline, Region.

Возврат: nil.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 0c32cef4>
```

```
(setq obj (vla-AddHatch obj acHatchPatternTypePreDefined
    "ANGLE" :vlax-true AcHatchObject))
;;; #<VLA-OBJECT IAcadHatch 0c32ce44>
(setq temp (list (vlax-ename->vla-object (car (entsel)))))
;;; (#<VLA-OBJECT IAcadLWPolyline 0c32db54>)
(setq temp (vlax-safearray-fill (vlax-make-safearray vlax-vbObject
    '(0 . 0)) temp))
;;; #<safearray...>
(vla-AppendOuterLoop obj temp)
```

7.2.69. Метод *AppendVertex*

Объекты: **3DPolyline**, **Polyline**, **PolygonMesh**.

Описание: добавление вершин.

Аргументы:

- <массив вершин> — координаты трехмерной точки (для PolygonMesh — одномерный массив координат точек) в виде безопасного массива.

Примечание

Размерность массива должна соответствовать размерности объекта.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadPolygonMesh 0c4399f4>
(vla-get-NVertexCount obj) ;_ выясняем количество точек массива
;;; 3
(vla-AppendVertex obj (vlax-safearray-fill (vlax-make-safearray
  vlax-vbDouble '(0 . 8)) (append (getpoint) (getpoint) (getpoint))))
```

7.2.70. Метод *ArrayPolar*

Объекты: **все графические объекты**.

Описание: размножение объектов в форме кругового массива.

Аргументы:

- <количество элементов> — положительное целое;
- <сектор заполнения> — угол в радианах, вещественное число, где положительное значение соответствует заполнению против часовой стрелки;
- <центр> — координаты трехмерной точки в виде безопасного массива.

Возврат: вариант (массив) созданных объектов.

Примечание

Метод выполняется с поворотом копий.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadLWPolyline 0c2bf5a4>
(setq temp (vla-ArrayPolar obj 3 (* pi 1.5) (vlax-3D-point (getpoint))))
;;; #<variant 8201 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (#<VLA-OBJECT IAcadLWPolyline 0c2e61b4>
;;;  #<VLA-OBJECT IAcadLWPolyline 0c2e60d4>)
```

7.2.71. Метод *ArrayRectangular*

Объекты: **все графические объекты**.

Описание: размножение объектов в форме трехмерного прямоугольного массива.

Аргументы:

- <строк>, <колонок>, <уровней> — положительные целые числа;
- <строки>, <колонки>, <уровни> — вещественные числа, расстояния.

Возврат: вариант (массив) созданных объектов.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadLWPolyline 0c2e61b4>
(setq temp (vla-ArrayRectangular obj 3 1 2 60 0 20))
;;; #<variant 8201 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (#<VLA-OBJECT IAcadLWPolyline 0c2e8294>
;;; #<VLA-OBJECT IAcadLWPolyline 0c2e81a4> ...)
```

7.2.72. Метод *AttachExternalReference*

Объекты: **ModelSpace**, **PaperSpace**, **Block**.

Описание: создание объекта ExternalReference.

Аргументы:

- <файл> — строка, путь к присоединяемому файлу;
- <имя> — строка, имя внешней ссылки;
- <точка вставки> — координаты трехмерной точки в виде варианта с типом безопасного массива;
- <масштаб X>, <масштаб Y>, <масштаб Z> — масштабные коэффициенты по осям, вещественные числа;
- <угол поворота> — вещественное число, угол в радианах;

- <признак наследования>:

  `:vlax-true` — не наследуемая;

  `:vlax-false` — наследуемая.
- [<пароль>] — строка.

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 0feec2c4>
(vla-AttachExternalReference obj "D:/Tu ... w/MyRef.dwg" "TestRef"
    (vlax-3D-point (getpoint)) 1.0 1.0 1.0 0.0 :vlax-true)
;;; #<VLA-OBJECT IAcadExternalReference 0ff77db4>
```

7.2.73. Метод *AttachToolbarToFlyout*

Объект: **ToolbarItem**.

Описание: связывание панели с групповой кнопкой другой панели.

Аргументы:

- <меню> — строка (имя меню, содержащего связываемую панель);
- <панель> — строка, имя панели инструментов.

Возврат: `nil`.

```
(setq toolbars (vla-get-Toolbars
    (vla-Item (vla-get-MenuGroups (vlax-get-acad-object)) "ACAD")))
;;; #<VLA-OBJECT IAcadToolbars 0c44ce94>
(setq tb1 (vla-Add toolbars "Include"))
;;; #<VLA-OBJECT IAcadToolbar 0c44c9fc>
;;; Создали панель, которую будем присоединять.
(setq but (vla-AddToolbarButton tb1 0 "Simple" "Help"
 "\003\003test "))
;;; #<VLA-OBJECT IAcadToolbarItem 0112ab0c>
;;; Создали простую кнопку в присоединяемой панели, чтобы была.
(setq tb2 (vla-Add toolbars "Container"))
;;; #<VLA-OBJECT IAcadToolbar 0c44068c>
;;; Создали панель, к которой будем присоединять.
(setq obj (vla-AddToolbarButton tb2 0 "Flyout" "Help" " "
 :vlax-true))
;;; #<VLA-OBJECT IAcadToolbarItem 0c4650bc>
;;; Создали групповую кнопку в панели-контейнере.
(vla-AttachToolbarToFlyout obj "acad" "Include")
;;; Присоединили панель. Получили вложенную панель с одной кнопкой.
(vla-AttachToolbarToFlyout obj "acad" "draw")
;;; Присоединили стандартную панель рисования на то же место.
(vla-Delete tb2) (vla-Delete tb1)
;;; Прибрались после экспериментов.
```

7.2.74. Метод *AuditInfo*

Объект: **Document**.

Описание: проверка целостности структуры документа.

Аргументы:

- <признак автокоррекции>:

  :vlax-true — автоматически исправлять ошибки;

  :vlax-false — спрашивать пользователя.

Возврат: nil.

```
(setq obj (vla-Item (vla-get-Documents (vlax-get-acad-object)) 1))
;;; #<VLA-OBJECT IAcadDocument 0c46df40>
(vla-AuditInfo obj :vlax-true)
```

7.2.75. Метод *Bind*

Объект: **Block**.

Описание: внедрение внешней ссылки.

Аргументы:

- <префикс имен>:

  :vlax-true — не использовать префиксы;

  :vlax-false — добавлять префикс вида <blockname>x.

Возврат: nil.

```
(setq obj (vla-Item (vla-get-Blocks actdoc) "TestRef"))
;;; #<VLA-OBJECT IAcadBlock2 0c4666a4>
(vla-Bind obj :vlax-true)
```

7.2.76. Метод *Block*

Объект: **SortentsTable**.

Описание: получение указателя на блок таблицы управления порядком следования объектов рисунка.

Аргументы: нет. Возврат: VLA-объект.

```
(setq dict (vla-GetExtensionDictionary (vla-get-ModelSpace actdoc)))
;;; #<VLA-OBJECT IAcadDictionary 0c3da5a4>
(setq obj (vla-AddObject dict "ACAD_SORTENTS" "AcDbSortentsTable"))
;;; #<VLA-OBJECT IAcadSortentsTable 0c3da4b4>
(vla-Block obj)
;;; #<VLA-OBJECT IAcadModelSpace2 0c3d9434>
```

7.2.77. Метод *Boolean*

Объекты: **3DSolid**, **Region**.

Описание: выполнение булевых операций.

Аргументы:

- <операция> — тип операции:

  acUnion (0) — объединение;

  acIntersection (1) — пересечение;

  acSubtraction (2) — вычитание.
- <объект> — VLA-объект второго примитива, участвующего в операции.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcad3DSolid 0be98a84>
(setq temp (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcad3DSolid 0be98664>
(vla-Boolean obj acSubtraction temp)
```

7.2.78. Метод *CheckInterference*

Объект: **3DSolid**.

Описание: проверка пересечения двух тел.

Аргументы:

- <объект> — VLA-объект (3DSolid);
- <признак создания>:

  :vlax-true — создать тело пересечения из общей части;

  :vlax-false — не создавать тело пересечения.

Возврат: созданный VLA-объект или nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcad3DSolid 0f59faa4>
(setq temp (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcad3DSolid 0f59c3f4>
(vla-CheckInterference obj temp :vlax-true)
;;; #<VLA-OBJECT IAcad3DSolid 0f59a9e4>
```

7.2.79. Метод *Clear*

Объекты: **SelectionSet**, **KeyValues**.

Описание: очистка набора.

Примечание

Для `KeyValues` метод является устаревшим, сохранен для совместимости.

Аргументы: нет. Возврат: `nil`.

```
(setq obj (vla-get-ActiveSelectionSet actdoc))
;;; #<VLA-OBJECT IAcadSelectionSet 0f5a2494>
(vla-Clear obj)
```

7.2.80. Метод *ClearSubSelection*

2005

Объект: **`Table`**.

Описание: удаление выборки ячеек из таблицы.

Аргументы: нет. Возврат: `nil`.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0f5a1a44>
(vla-ClearSubSelection obj)
```

7.2.81. Метод *ClearTableStyleOverrides*

2005

Объект: **`Table`**.

Описание: очистка таблицы от наложенного форматирования.

Аргументы:

- `<флаг>` — режим очистки:

  `0` — очистка таблицы и ячеек;

  `1` — очистка только таблицы;

  `2` — очистка только ячеек.

Возврат: `nil`.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0f5a1a44>
(vla-ClearTableStyleOverrides obj 0)
```

7.2.82. Метод *ClipBoundary*

Объект: **`Raster`**.

Описание: формирование линии подрезки.

Аргументы:

- `<точки>` — вариант (плоский безопасный массив), содержащий координаты двумерных точек (не менее двух значащих).

Примечание

Первая и последняя точки должны совпадать, таким образом имитируется замыкание контура, а последняя точка является незначащей. Кроме того, если хотя бы одна из точек обрезки задана вне растра, то изображение самого растра не выводится на экран (указано Э. Смолянкой). Поэтому передаваемые точки контура подрезки необходимо предварительно проверять на принадлежность прямоугольнику растрового объекта.

Возврат: `nil`.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadRasterImage 0f5c7924>
(setq temp (list (getpoint) (getpoint) (getpoint)))
;;; ((657.871 446.653 0.0) (537.573 840.826 0.0) (1034.81 621.692 0.0))
;;; создали список из трех трехмерных точек
(setq temp (mapcar 'list (mapcar 'car temp) (mapcar 'cadr temp)))
;;; ((657.871 446.653) (537.573 840.826) (1034.81 621.692))
;;; преобразовали список точек к двумерному
(setq temp (cons (last temp) temp))
;;; ((1034.81 621.692) (657.871 446.653) (537.573 840.826)
;;;  (1034.81 621.692))
;;; повторили последнюю точку в начале списка точек
(setq temp (apply 'append temp))
;;; (1034.81 621.692 657.871 446.653 537.573 840.826 1034.81 621.692)
;;; преобразовали список в плоский
(setq temp (vlax-make-variant (vlax-safearray-fill
    (vlax-make-safearray vlax-vbDouble '(0 . 7)) temp)))
;;; #<variant 8197 ...>
;;; создали массив координат
(vla-ClipBoundary obj temp)
```

7.2.83. Метод *Close*

Объекты: **Document**, **Documents**.

Описание: закрытие документа или всех документов.

Аргументы:

- `[<признак сохранения>]`:

  `:vlax-true` — сохранять;

  `:vlax-false` — не сохранять.
- `[<имя>]` — строка, новое имя документа.

Примечание

Аргументы применимы только при закрытии одиночного документа. Учитываются только при наличии несохраненных изменений. При сохранении документа, который ранее не сохранялся и имя ему не задано, оно будет назначено из свойства `Name`, а путь из свойства **Path** сохраняемого документа.

Возврат: `nil`.

Примечание

Закрытие активного документа средствами AutoLISP невозможно. Применение данного метода к коллекции (семейству) приведет к закрытию всех незанятых документов, кроме активного, с сообщением об ошибке автоматизации. Перехват ошибки можно выполнить с помощью функции `vl-catch-all-apply`.

```
(setq obj (vla-Item (vla-get-Documents (vlax-get-acad-object)) "Tt.dwg"))
;;; #<VLA-OBJECT IAcadDocument 0f5bc930>
(vla-Close obj :vlax-true)
;;; закрываем неактивный документ с сохранением изменений
(setq obj (vla-get-Documents (vlax-get-acad-object)))
;;; #<VLA-OBJECT IAcadDocuments 0f5febc0>
(vla-Close obj)
;;; error: Automation Error. Drawing is busy.
;;; при попытке закрытия всех документов получаем сообщение об ошибке
```

7.2.84. Метод *Connect*

Объект: **DbConnect**.

Описание: подключение к источнику данных.

Аргументы:

- `<источник>` — строка, имя UDL-файла;
- `[<пользователь>]` — строка, имя пользователя;
- `[<пароль>]` — строка;
- `[<сообщение>]` — запрос пользователя о параметрах подключения, при необходимости:

  `:vlax-true` — запрашивать;

  `:vlax-false` — не запрашивать.

Возврат: `nil`.

```
(setq DBConn (vla-GetInterfaceObject (vlax-get-acad-object)
               "CAO.DbConnect.16"))
;;; #<VLA-OBJECT IDbConnect 05958a60>
(vlax-invoke DBConn 'Connect "jet_dbsamples")
```

7.2.85. Метод *ConvertToAnonymousBlock*

Объекты: **BlockRef**, **MInsertBlock**.

Описание: конвертация динамического блока в статический анонимный.

Аргументы: нет. Возврат: `nil`.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadBlockReference2 0b100d24>
(vla-ConvertToAnonymousBlock obj)
```

7.2.86. Метод *ConvertToStaticBlock*

2006

Объекты: **BlockRef**, **MInsertBlock**.

Описание: конвертация динамического блока в статический с именем.

Аргументы:

- <имя> — строка.

Возврат: `nil`.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadBlockReference2 0b100d24>
(vla-ConvertToStaticBlock obj "NewBlock")
```

7.2.87. Метод *Copy*

Объекты: **все графические объекты**.

Описание: копирование объектов.

Аргументы: нет.

Возврат: созданный VLA-объект.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadLWPolyline 0c280e04>
(vla-Copy obj)
;;; #<VLA-OBJECT IAcadLWPolyline 0c2819d4>
```

7.2.88. Метод *CopyFrom*

Объекты: **DimStyle**, **PlotConfiguration**.

Описание: копирование параметров одного стиля (конфигурации) в другой.

Аргументы:

- <исходный объект> — объект соответствующего типа.

Возврат: `nil`.

```
(setq obj (vla-Item (vla-get-DimStyles actdoc) "ESKD"))
;;; #<VLA-OBJECT IAcadDimStyle 0c27d0a4>
```

```
(setq temp (vla-Item (vla-get-DimStyles actdoc) "ISO-25"))
;;; #<VLA-OBJECT IAcadDimStyle 0c27d054>
(vla-CopyFrom obj temp)
```

7.2.89. Метод *CopyObjects*

Объекты: **Database**, **Document**.

Описание: клонирование объектов в другой документ, блок или пространство.

Аргументы:

- <объекты> — вариант (безопасный массив объектов);
- [<владелец>] — указатель на VLA-объект места назначения;
- [<пары клонирования>] — указатель на переменную, в которой будет сохранен массив пар клонирования.

Возврат: массив созданных объектов.

```
(setq actsel (vla-get-ActiveSelectionSet actdoc) tmp nil)
(vla-Clear actsel)
(vla-SelectOnScreen actsel)
(setq block (vla-Add (vla-get-Blocks actdoc)
    (vlax-3D-point (getpoint)) "TB"))
;;; #<VLA-OBJECT IAcadBlock2 0c29cd14>
(setq temp (vla-CopyObjects actdoc (vlax-make-variant
  (vlax-safearray-fill (vlax-make-safearray vlax-vbObject
     (cons 0 (1- (vla-get-Count actsel))))
  (vlax-for x actsel (setq tmp (cons x tmp))))) block 'idpairs))
;;; #<variant 8201 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (#<VLA-OBJECT IAcadLWPolyline 0c29fa84> #<VLA-OBJECT ...)
(vlax-safearray->list idpairs)
;;; (#<VLA-OBJECT IAcadIdPair 0c29f954> #<VLA-OBJECT ...)
```

См. также *приложение 8*.

7.2.90. Метод *CopyProfile*

Объект: **PreferencesProfiles**.

Описание: создание нового профиля путем копирования существующего.

Аргументы:

- <исходный>, <новый> — строковые, имена профилей.

Возврат: nil.

```
(setq obj (vla-get-Profiles
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesProfiles 0c29f10c>
(vla-CopyProfile obj "<<Unnamed Profile>>" "My Profile")
```

7.2.91. Метод *CreateEntry*

Объект: **FileDependencies**.

Описание: создание новой записи в таблице зависимостей.

Аргументы:

- <признак> — строка, описание приложения или особенностей;
- <файл> — строка, полное имя файла;
- <признак влияния на графику>:

  :vlax-true — влияет;

  :vlax-false — не влияет.
- <признак неувеличения счетчика ссылок>:

  :vlax-true — не увеличивать;

  :vlax-false — увеличивать.

Возврат: целое число, счетчик в таблице зависимостей.

```
(setq obj (vla-get-FileDependencies actdoc))
;;; #<VLA-OBJECT IAcadFileDependencies 0beb2514>
(vla-CreateEntry obj "Acad:Image" "C:\\Doc ... es\\Window.jpg"
 :vlax-true :vlax-false)
;;; 10
```

7.2.92. Метод *CreateLabel*

Объект: **LinkTemplate**.

Описание: создание связанного или свободного ярлыка.

Аргументы:

- <аргумент1> — положительное целое число, ID объекта, к которому будет привязан ярлык, или координаты трехмерной точки в виде варианта с типом безопасного массива (точка вставки для свободного ярлыка);
- <значения ключей> — VLA-объект (KeyValues);

Примечание

Порядок создания, количество и содержимое ключей должно соответствовать используемому шаблону связи. При отсутствии указанных значений ключей в связываемой таблице связь и ярлык создаются, но связывания не происходит.

- <шаблон ярлыка> — VLA-объект (LabelTemplate) или строка с именем шаблона;

Примечание

Шаблон ярлыка должен соответствовать шаблону связи.

- [<отображаемые значения>] — вариант (безопасный массив), содержащий значения, отображаемые при отсутствии значащих ключей в базе.

Возврат: созданный VLA-объект.

```
(setq DBConn (vla-GetInterfaceObject (vlax-get-acad-object)
               "CAO.DbConnect.16"))
;;; #<VLA-OBJECT IDbConnect 0c1cc990>
(setq DBLinksT (vlax-invoke DBConn 'GetLinkTemplates))
;;; #<VLA-OBJECT ILinkTemplates 0c1ccb10>
(setq DBLinkT (vla-Item DBLinksT "Link2"))
;;; #<VLA-OBJECT ILinkTemplate 0c1d0770>
(setq DBLablsT (vlax-invoke DBConn 'GetLabelTemplates))
;;; #<VLA-OBJECT ILabelTemplates 0c1cc7c0>
(setq DBLablT (vla-Item DBLablsT "Label4"))
;;; #<VLA-OBJECT ILabelTemplate 0c1f34b0>
(setq KeyVals (vla-GetInterfaceObject (vlax-get-acad-object)
    "CAO.KeyValues.16"))
;;; #<VLA-OBJECT IKeyValues 0c1cdf80>
(vlax-invoke KeyVals 'Add "MANUFACTURER" "HEWLETT PACKARD")
;;; #<VLA-OBJECT IKeyValue 0c1f5c40>
(vlax-invoke KeyVals 'Add "Equipment_Description" "LASERJET 2")
;;; #<VLA-OBJECT IKeyValue 0c1f6570>
(vlax-invoke DBLinkT 'CreateLabel (vla-get-ObjectID
    (vlax-ename->vla-object (car (entsel)))) KeyVals DBLablT)
;;; #<VLA-OBJECT ILabel 0c1f3c10>
```

7.2.93. Метод *CreateLink*

Объект: **LinkTemplate**.

Описание: создание связи без ярлыка.

Аргументы:

- <аргумент1> — положительное целое число, ID связываемого объекта;
- <значения ключей> — VLA-объект, KeyValues.

Примечание

Порядок создания, количество и содержимое ключей должно соответствовать используемому шаблону связи. При отсутствии указанных значений ключей в связываемой таблице связь создается, но само связывание не происходит.

Возврат: созданный VLA-объект.

```
(setq DBConn (vla-GetInterfaceObject (vlax-get-acad-object)
               "CAO.DbConnect.16"))
;;; #<VLA-OBJECT IDbConnect 0c1cc990>
(setq DBLinksT (vlax-invoke DBConn 'GetLinkTemplates))
;;; #<VLA-OBJECT ILinkTemplates 0c1ccb10>
(setq DBLinkT (vla-Item DBLinksT "Link2"))
;;; #<VLA-OBJECT ILinkTemplate 0c1d0770>
(setq KeyVals (vla-GetInterfaceObject (vlax-get-acad-object)
    "CAO.KeyValues.16"))
;;; #<VLA-OBJECT IKeyValues 0c1cdf80>
(vlax-invoke KeyVals 'Add "MANUFACTURER" "HEWLETT PACKARD")
;;; #<VLA-OBJECT IKeyValue 0c1f5c40>
(vlax-invoke KeyVals 'Add "Equipment_Description" "LASERJET 2")
;;; #<VLA-OBJECT IKeyValue 0c1f6570>
(vlax-invoke DBLinkT 'CreateLink (vla-get-ObjectID
    (vlax-ename->vla-object (car (entsel)))) KeyVals)
;;; #<VLA-OBJECT ILink 0c1f1c10>
```

7.2.94. Метод *CreateTypedArray*

Объект: `Utility`.

Описание: создание безопасного массива с автоматическим определением размера (не менее 2) и приведением типов (если возможно). Используется при позднем связывании.

Аргументы:

- `<массив>` — указатель на переменную;
- `<тип>`:

  `vlax-vbInteger` (2) — целое;

  `vlax-vbLong` (3) — длинное целое;

  `vlax-vbSingle` (4) — вещественное одинарной точности;

  `vlax-vbDouble` (5) — вещественное двойной точности;

  `vlax-vbString` (8) — строковое;

  `vlax-vbObject` (9) — объект;

  `vlax-vbBoolean` (11) — логическое;

  `vlax-vbVariant` (12) — вариант.
- `<значение1>`, `<значение2>`, `[<значениеN>]` — значения, которыми будет заполнен вариант.

Возврат: `nil`.

```
(setq obj (vla-get-Utility actdoc))
;;; #<VLA-OBJECT IAcadUtility 0beb2dd4>
(vla-CreateTypedArray obj 'temp vlax-vbString 1 "два" 3.21)
(vlax-safearray->list temp)
;;; ("1" "два" "3.21")
```

7.2.95. Метод *Delete*

Объекты: **все графические объекты**, **Block**, **Dictionary**, **DimStyle**, **Group**, **Hyperlink**, **LabelTemplates**, **Layer**, **LayerStateManager**, **Layout**, **Linetype**, **Link**, **LinkTemplates**, **PlotConfiguration**, **PopupMenuItem**, **PViewport**, **Queries**, **RegisteredApplication**, **SelectionSet**, **TextStyle**, **Toolbar**, **ToolbarItem**, **UCS**, **View**, **Viewport**, **XRecord**.

Описание: удаление объектов.

Аргументы:

- <имя> — строка с именем удаляемого состояния (только для LayerStateManager), или строка с именем удаляемого объекта в коллекции (только для LabelTemplates, LinkTemplates, Queries), или целое число — индекс в коллекции (семействе).

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadLWPolyline 0beb1114>
(vla-Delete obj)
```

7.2.96. Метод *DeleteCellContent*

Объект: **Table**.

Описание: удаление содержимого ячейки.

Аргументы:

- <строка>, <колонка> — неотрицательные целые, отсчет с нуля.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0beb2cb4>
(vla-DeleteCellContent obj 1 2)
```

7.2.97. Метод *DeleteColumns*

Объект: **Table**.

Описание: удаление колонок.

Аргументы:

- <начальная колонка> — неотрицательное целое, отсчет с нуля;
- <количество> — положительное целое, количество удаляемых колонок.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0beb2cb4>
(vla-DeleteColumns obj 0 1)
```

7.2.98. Метод *DeleteConfiguration*

Объект: **Viewports**.

Описание: удаление именованной конфигурации.

Аргументы:

- <имя> — строка, имя удаляемой конфигурации.

Возврат: nil.

```
(setq obj (vla-get-Viewports actdoc))
;;; #<VLA-OBJECT IAcadViewports 0bebd314>
(vla-DeleteConfiguration obj "test")
```

7.2.99. Метод *DeleteFitPoint*

Объект: **Spline**.

Описание: удаление опорной точки.

Аргументы:

- <индекс> — неотрицательное целое, отсчет с нуля.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadSpline 0bead764>
(vla-DeleteFitPoint obj 1)
```

7.2.100. Метод *DeleteProfile*

Объект: **PreferencesProfiles**.

Описание: удаление профиля.

Аргументы:

- <имя> — строковое, имя удаляемого профиля.

Возврат: nil.

```
(setq obj (vla-get-Profiles
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesProfiles 0bec15ec>
(vla-DeleteProfile obj "My Profile")
```

7.2.101. Метод *DeleteRows*

2005

Объект: **Table**.

Описание: удаление строк.

Аргументы:

- <начальная строка> — неотрицательное целое, отсчет с нуля;
- <количество> — положительное целое, количество удаляемых строк.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0beb2cb4>
(vla-DeleteRows obj 1 1)
```

7.2.102. Метод *Detach*

Объект: **Block**.

Описание: отсоединение внешней ссылки (внешние ссылки размещены в таблице блоков).

Аргументы: нет. Возврат: nil.

```
(setq obj (vla-Item (vla-get-Blocks actdoc) "MyTestRef"))
;;; #<VLA-OBJECT IAcadBlock2 05848014>
(vla-Detach obj)
```

7.2.103. Метод *Disconnect*

Объект: **DbConnect**.

Описание: отключение источника данных.

Аргументы:

- <источник> — строка, имя UDL-файла.

Возврат: nil.

```
(setq DBConn (vla-GetInterfaceObject (vlax-get-acad-object)
              "CAO.DbConnect.16"))
;;; #<VLA-OBJECT IDbConnect 0c1cc990>
(vlax-invoke DBConn 'Disconnect "jet_dbsamples")
```

7.2.104. Метод *Display*

Объект: **PViewport**.

Описание: переключение режимов видового экрана.

Аргументы:

- <режим>:

  :vlax-true — включен;

  :vlax-false — выключен.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadPViewport2 0becf514>
(vla-Display obj :vlax-false)
```

7.2.105. Метод *DisplayPlotPreview*

Объект: **Plot**.

Описание: предварительный просмотр.

Аргументы:

- <режим отображения>:

  acPartialPreview (0) — контурное;

  acFullPreview (1) — детальное.

Примечание

Ввиду того, что версия R16 поддерживает только режим полного (детального) предварительного просмотра печати, выбор любого из указанных значений аргумента в системе AutoCAD 2004—2006 равнозначен.

Возврат: nil.

```
(setq obj (vla-get-Plot actdoc))
;;; #<VLA-OBJECT IAcadPlot 0beee36c>
(vla-DisplayPlotPreview obj acFullPreview)
```

7.2.106. Метод *DistanceToReal*

Объект: **Utility**.

Описание: преобразование значения из строкового представления в вещественное.

Аргументы:

- <исходное> — строковое представление числа;
- <единицы> — входной формат представления значения:

  acDefaultUnits (–1) — по умолчанию;

  acScientific (1) — научные;

  acDecimal (2) — десятичные;

  acEngineering (3) — инженерные;

  acArchitectural (4) — архитектурные;

  acFractional (5) — дробные.

Возврат: вещественное число.

```
(setq obj (vla-get-Utility actdoc))
;;; #<VLA-OBJECT IAcadUtility 0beb2dd4>
(vla-DistanceToReal obj "123" acDecimal)
;;; 123.0
```

7.2.107. Метод *Dock*

Объект: **Toolbar**.

Описание: фиксация панели у соответствующего края графического окна.

Аргументы:

- <сторона> — указание стороны, с которой будет зафиксирована панель:

  acToolbarDockTop (0) — верхняя;

  acToolbarDockBottom (1) — нижняя;

  acToolbarDockLeft (2) — левая;

  acToolbarDockRight (3) — правая.

Возврат: nil.

```
(setq obj (vla-Item (vla-get-Toolbars (vla-Item (vla-get-MenuGroups
    (vlax-get-acad-object)) "acad")) "Viewports"))
;;; #<VLA-OBJECT IAcadToolbar 010a71dc>
(vla-Dock obj acToolbarDockTop)
```

7.2.108. Метод *ElevateOrder*

Объект: **Spline**.

Описание: повышение порядка сплайна.

Аргументы:

- <порядок> — положительное целое.

Примечание

Минимальное значение на 1 больше значения свойства Degree.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadSpline 0bef9ad4>
(vla-get-Degree obj)
;;; 3
(vla-ElevateOrder obj 5)
(vla-get-Degree obj)
;;; 4
```

7.2.109. Метод *EndUndoMark*

Объект: **Document**.

Описание: установка конечной метки отката. См. метод StartUndoMark.

Аргументы: нет. Возврат: nil.

```
(vla-EndUndoMark actdoc)
```

7.2.110. Метод *Erase*

Объект: **SelectionSet**.

Описание: стирание графических объектов включенных в набор.

Аргументы: нет. Возврат: nil.

```
(setq obj (vla-get-ActiveSelectionSet actdoc))
;;; #<VLA-OBJECT IAcadSelectionSet 0bef85e4>
(vla-Clear obj)
(vla-SelectOnScreen obj)
(vla-Erase obj)
```

7.2.111. Метод *Eval*

Объект: **Application**.

Описание: выполнение выражения языка VBA.

Аргументы:

- <выражение> — строка.

Возврат: `nil`.

```
(setq obj (vlax-get-acad-object))
;;; #<VLA-OBJECT IAcadApplication 00b9d604>
(vla-Eval obj "MsgBox  \"Информация о ...\", vbCritical, \"Проблема\"")
```

7.2.112. Метод *Evaluate*

Объекты: **Hatch**, **Leader**.

Описание: пересчет геометрии объекта.

Аргументы: нет. Возврат: `nil`.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadHatch 0b8d2e54>
(vla-put-PatternScale obj 0.75)
(vla-Evaluate obj)
(vla-Update obj)
```

7.2.113. Метод *Explode*

Объекты: **3DPolyline**, **BlockRef**, **LightweightPolyline**, **Polygonmesh**, **Polyline**, **Region**.

Описание: расчленение копии составного объекта.

Примечание

Исходный объект не удаляется. В разных версиях AutoCAD этот метод выполняется неидентично. Особенно неудачна его реализация в версии 2004, где происходит смещение копий расчлененных объектов относительно их положения во вставке блока.

Аргументы: нет.

Возврат: массив созданных VLA-объектов в виде варианта.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadBlockReference 0b9059e4>
(setq temp (vla-Explode obj))
;;; #<variant 8201 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (#<VLA-OBJECT IAcadCircle 0b905c74> #<VLA-OBJECT IAcad ...)
```

7.2.114. Метод *Export*

Объекты: **Document**, **LayerStateManager**.

Описание: экспорт во внешний файл примитивов или состояний слоев.

Аргументы (различные случаи):

- **Document** (создается пустой блок)
  - <файл> — строка, имя файла;
  - <формат> — строка, расширение файла, ассоциированное с форматом (регистр не учитывается). Допустимые значения: "wmf", "sat", "eps", "dsf", "bmp";
  - <набор> — набор графических объектов для экспорта.
- **LayerStateManager**
  - <состояние> — имя сохраненного состояния слоев;
  - <файл> — строка, имя файла.

Возврат: nil.

```
(setq temp (vla-get-ActiveSelectionSet actdoc))
;;; #<VLA-OBJECT IAcadSelectionSet 0bf1a264>
(vla-Clear temp)
(vla-SelectOnScreen temp)
(vla-Export actdoc "d:/temp/test" "wmf" temp)
;;; Экспортировали примитивы во внешний файл.
(setq obj (vla-GetInterfaceObject (vlax-get-acad-object)
    "AutoCAD.AcadLayerStateManager.16"))
;;; #<VLA-OBJECT IAcadLayerStateManager 127f86c4>
(vla-SetDatabase obj (vla-get-Database (vla-get-ModelSpace actdoc)))
;;; Инициализация объекта.
(vla-Export obj "MyState" "D:/Temp/ExState.las")
;;; Экспортировали предварительно сохраненное состояние слоев.
```

7.2.115. Метод *ExportProfile*

Объект: **PreferencesProfiles**.

Описание: экспорт профиля во внешний файл.

Аргументы:

- <профиль> — строка, имя профиля;
- <файл> — строка, имя файла с расширением arg.

Возврат: nil.

```
(setq obj (vla-get-Profiles
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesProfiles 0bf1a94c>
(vla-ExportProfile obj "<<Unnamed Profile>>" "d:/temp/test.arg")
```

7.2.116. Метод *FieldCode*

Объекты: **MText**, **Text**.

Описание: получение внутреннего содержимого текстовой строки, содержащей параметрические поля.

Аргументы: нет. Возврат: строка.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadText2 0bf1a764>
(vla-FieldCode obj)
;;; "Площадь круга %<\\AcObjProp Object(%<\\_ObjId 2130054576>%).Area>%"
;;; В примитиве Text использовано поле с вычисляемой площадью
;;; другого объекта (Circle, ObjId = 2130054576).
```

7.2.117. Метод *Float*

Объект: **Toolbar**.

Описание: перевод панели инструментов в плавающее состояние.

Аргументы:

- <слева>, <сверху> — положительные целые числа, отступ в пикселах;
- <строк> — положительное целое число, количество строк.

Возврат: nil.

```
(setq obj (vla-Item (vla-get-Toolbars (vla-Item (vla-get-MenuGroups
    (vlax-get-acad-object)) "acad")) "Modify"))
;;; #<VLA-OBJECT IAcadToolbar 010cf5fc>
(vla-Float obj 150 100 2)
```

7.2.118. Метод *GenerateLayout*

Объект: **Table**.

Описание: приведение таблицы в соответствие с текущим стилем.

Примечание

Функционирование не отслеживается.

Аргументы: нет. Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0b70ef74>
(vla-GenerateLayout obj)
```

7.2.119. Метод *GenerateUsageData*

Объект: **Layers**.

Описание: генерирование специального значка у неиспользованных слоев в диалоговом окне **Layer Properties Manager** (Диспетчер свойств слоев). Соответствует установке значения 1 системной переменной SHOWLAYERUSAGE.

Аргументы: нет. Возврат: `nil`.

```
(setq obj (vla-get-Layers actdoc))
;;; #<VLA-OBJECT IAcadLayers2 0b8d4b94>
(vla-GenerateUsageData obj)
```

7.2.120. Метод *GetAcadState*

Объект: **Application**.

Описание: получение объекта состояния системы AutoCAD при вызове из другого приложения.

Аргументы: нет.

Возврат: VLA-объект.

```
(setq obj (vlax-get-acad-object))
;;; #<VLA-OBJECT IAcadApplication 00b9d604>
(vla-GetAcadState obj)
;;; #<VLA-OBJECT IAcadState 0b8e8ff0>
```

7.2.121. Метод *GetAlignment*

Объекты: **Table**, **TableStyle**.

Описание: получение типа выравнивания содержимого ячеек таблицы.

Аргументы:

- `<тип строк>` — указание типа строк:

  `acUnknownRow` (0) — не определено;

  `acDataRow` (1) — ячейки данных;

  `acTitleRow` (2) — ячейка заголовка таблицы;

  `acHeaderRow` (4) — ячейки заголовков столбцов.

Возврат: целое число. Интерпретацию см. метод `SetAlignment`.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0b8e8874>
(vla-GetAlignment obj acDataRow)
;;; 2
```

7.2.122. Метод *GetAllProfileNames*

Объект: **PreferencesProfiles**.

Описание: получение имен всех существующих профилей.

Аргументы:

- <переменная> — символ, имя переменной, которой будет присвоено значение.

Возврат: nil.

```
(setq obj (vla-get-Profiles
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesProfiles 0b8e9ddc>
(vla-GetAllProfileNames obj 'temp)
(vlax-safearray->list temp)
;;; ("<<Unnamed Profile>>" "Arch" ...)
```

7.2.123. Метод *GetAngle*

Объект: **Utility**.

Описание: интерактивное вычисление угла, в радианах, наклона вектора (заданного двумя точками) от нулевого направления, установленного системной переменной ANGBASE (с учетом предшествующего вызова метода InitializeUserInput).

Аргументы:

- [<точка>] — координаты первой трехмерной точки в виде варианта с типом безопасного массива;
- [<подсказка>] — строка.

Примечание

При отсутствии аргумента <точка> для передачи подсказки требуется указать первый аргумент в виде nil.

Возврат: положительное вещественное число.

```
(setq obj (vla-get-Utility actdoc))
;;; #<VLA-OBJECT IAcadUtility 0b8e9c84>
(vla-GetAngle obj nil "\nКликни пару раз: ")
;;; 0.588607
```

7.2.124. Метод *GetAttachmentPoint*

Объект: **Table**.

Описание: получение координат точки привязки содержимого ячейки.

Аргументы:

- <строка>, <колонка> — неотрицательные целые числа.

Возврат: координаты точки в виде варианта с типом безопасного массива.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0b8089d4>
(setq temp (vla-GetAttachmentPoint obj 2 1))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (2206.56 572.429 0.0)
```

7.2.125. Метод *GetAttributes*

Объекты: **BlockRef**, **ExternalReference**, **MInsertBlock**.

Описание: получение массива атрибутов.

Аргументы: нет.

Возврат: массив атрибутов в виде варианта с типом безопасного массива.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadMInsertBlock 0b808fa4>
(setq temp (vla-GetAttributes obj))
;;; #<variant 8201 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (#<VLA-OBJECT IAcadAttributeReference 0b80aa94>)
```

7.2.126. Метод *GetAutoScale*

Объект: **Table**.

Описание: получение значения режима автовыравнивания содержимого ячейки.

Аргументы:

- <строка>, <колонка> — положительные целые числа.

Возврат:

- :vlax-true — включен;
- :vlax-false — выключен.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0b8089d4>
(vla-GetAutoScale obj 2 1)
;;; :vlax-false
```

7.2.127. Метод *GetBackgroundColor*

2005

Объекты: **Table**, **TableStyle**.

Описание: получение значения цвета фона строк.

Аргументы:

- <тип строк> — указание типа строк, см. метод GetAlignment.

Возврат: объект типа AcCmColor.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0b8089d4>
(vla-GetBackgroundColor obj acDataRow)
;;; #<VLA-OBJECT IAcadAcCmColor 0b809c60>
```

7.2.128. Метод *GetBackgroundColorNone*

2005

Объекты: **Table**, **TableStyle**.

Описание: признак отсутствия цвета фона у строк.

Аргументы:

- <тип строк> — указание типа строк, см. метод GetAlignment.

Возврат:

- :vlax-true — включен (цветного фона нет);
- :vlax-false — выключен.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0b8089d4>
(vla-GetBackgroundColorNone obj acDataRow)
;;; :vlax-true
```

7.2.129. Метод *GetBitmaps*

Объект: **ToolbarItem**.

Описание: получение имен иконок кнопки.

Аргументы:

- <малая>, <большая> — символы, которым присваиваются строки.

Возврат: nil.

```
(setq obj (vla-Item (vla-Item (vla-get-Toolbars (vla-Item
 (vla-get-MenuGroups (vlax-get-acad-object)) "acad")) "TestToolbar") 0))
;;; #<VLA-OBJECT IAcadToolbarItem 0b825fbc>
(vla-GetBitmaps obj 'small 'large)
```

```
small
;;; "RCDA.bmp"
large
;;; "RCDATA_16_BLANK"
```

7.2.130. Метод *GetBlockAttributeValue*

2005

Объект: **Table**.

Описание: получение значения атрибута блока из ячейки.

Аргументы:

- <строка>, <колонка>, <ID атрибута> — неотрицательные целые числа.

Возврат: строка.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0b8089d4>
(vla-GetBlockAttributeValue obj 1 3 (vla-get-ObjectID
    (vla-Item (vla-Item (vla-get-Blocks actdoc) "ee") 2)))
;;; "test"
```

7.2.131. Метод *GetBlockRotation*

2005

Объект: **Table**.

Описание: получение значения угла, в радианах, поворота блока, вставленного в ячейку.

Аргументы:

- <строка>, <колонка> — неотрицательные целые числа.

Возврат: неотрицательное вещественное число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0b8089d4>
(vla-GetBlockRotation obj 1 3)
;;; 0.575959
```

7.2.132. Метод *GetBlockScale*

2005

Объект: **Table**.

Описание: получение значения масштаба блока из ячейки.

Аргументы:

- <строка>, <колонка> — неотрицательные целые числа.

Возврат: положительное вещественное число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0b8089d4>
(vla-GetBlockScale obj 1 3)
;;; 0.0143858
```

7.2.133. Метод *GetBlockTableRecordId*

Объект: **Table**.

Описание: получение идентификатора записи ячейки.

Аргументы:

- <строка>, <колонка> — неотрицательные целые числа.

Возврат: положительное целое число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0b8089d4>
(vla-GetBlockTableRecordId obj 1 3)
;;; 2130042424
```

7.2.134. Метод *GetBoundingBox*

Объекты: **все графические объекты**, **AttributeReference**.

Описание: получение двух диагональных точек для прямоугольника, описанного вокруг объекта.

Аргументы:

- <минимум>, <максимум> — символы, имена переменных, которым присваиваются значения координат в виде безопасного массива.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadCircle 0b919a54>
(vla-GetBoundingBox obj 'minp 'maxp)
(vlax-safearray->list minp)
;;; (2025.91 146.064 -1.0e-008)
(vlax-safearray->list maxp)
;;; (2280.27 400.42 1.0e-008)
```

7.2.135. Метод *GetBulge*

Объекты: **LightweightPolyline**, **Polyline**.

Описание: получение кривизны сегмента, актуально только для полилиний типа acSimplePoly.

Аргументы:

- <индекс> — неотрицательное целое, номер опрашиваемой вершины.

Возврат: вещественное число (¼ тангенса угла дуги), положительные значения соответствуют направлению против часовой стрелки.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadLWPolyline 0b9239b4>
(vla-GetBulge obj 3)
;;; 0.335171
```

7.2.136. Метод *GetCanonicalMediaNames*

Объекты: **Layout**, **PlotConfiguration**.

Описание: получение канонических имен форматов листов текущего принтера.

Аргументы: нет.

Возврат: массив строк в виде варианта.

```
(setq obj (vla-get-ActiveLayout actdoc))
;;; #<VLA-OBJECT IAcadLayout 0b9313b4>
(setq temp (vla-GetCanonicalMediaNames obj))
;;; #<variant 8200 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; ("Letter_(8.50_x_11.00_Inches)" "Legal ...)
```

7.2.137. Метод *GetCellAlignment*

Объект: **Table**.

Описание: получение типа выравнивания содержимого ячейки.

Аргументы:

- <строка>, <колонка> — неотрицательные целые числа.

Возврат: целое число, см. метод GetAlignment.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0b938224>
(vla-GetCellAlignment obj 2 2)
;;; 2
```

7.2.138. Метод *GetCellBackgroundColor*

Объект: **Table**.

Описание: получение цвета фона ячейки.

Аргументы:

- <строка>, <колонка> — неотрицательные целые числа.

Возврат: объект типа AcCmColor.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0b938224>
(vla-GetCellBackgroundColor obj 2 2)
;;; #<VLA-OBJECT IAcadAcCmColor 0b939a50>
```

7.2.139. Метод *GetCellBackgroundColorNone*

2005

Объект: **Table**.

Описание: признак отсутствия цветного фона у ячейки.

Аргументы:

- <строка>, <колонка> — неотрицательные целые числа.

Возврат:

- :vlax-true — включен (нет фона);
- :vlax-false — выключен.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0b938224>
(vla-GetCellBackgroundColorNone obj 2 2)
;;; :vlax-true
```

7.2.140. Метод *GetCellContentColor*

2005

Объект: **Table**.

Описание: получение цвета содержимого ячейки.

Аргументы:

- <строка>, <колонка> — неотрицательные целые числа.

Возврат: объект типа AcCmColor.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0b938224>
(vla-GetCellContentColor obj 2 2)
;;; #<VLA-OBJECT IAcadAcCmColor 0b9380a0>
```

7.2.141. Метод *GetCellExtents*

2005

Объект: **Table**.

Описание: получение координат ячейки с учетом отступов.

Аргументы:

- <строка>, <колонка> — неотрицательные целые числа;
- <направление>:

  :vlax-true — наружу;

  :vlax-false — внутрь.

Возврат: одномерный массив трехмерных координат четырех точек в виде варианта с типом безопасного массива.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0b938224>
(setq temp (vla-GetCellExtents obj 2 2 :vlax-false))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (2241.28 572.429 0.0 2304.72 572.429 0.0
;;;  2241.28 566.429 0.0 2304.72 566.429 0.0)
```

7.2.142. Метод *GetCellGridColor*

2005

Объект: **Table**.

Описание: получение цвета рамки ячейки.

Аргументы:

- <строка>, <колонка> — неотрицательные целые числа;
- <граница>:

  acTopMask (1) — верхняя;

  acRightMask (2) — правая;

  acBottomMask (4) — нижняя;

  acLeftMask (8) — левая.

Примечание

Значения могут складываться.

Возврат: объект типа AcCmColor.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0b938224>
(vla-GetCellGridColor obj 2 2 acBottomMask)
;;; #<VLA-OBJECT IAcadAcCmColor 0b941390>
```

7.2.143. Метод *GetCellGridLineWeight*

2005

Объект: **Table**.

Описание: получение веса линии границы ячейки.

Аргументы:

- <строка>, <колонка> — неотрицательные целые числа;
- <граница> — см. метод GetCellGridColor.

Возврат: см. свойство DimensionLineWeight.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0b938224>
(vla-GetCellGridLineWeight obj 2 2 acBottomMask)
;;; -2
```

7.2.144. Метод *GetCellGridVisibility*

2005

Объект: **Table**.

Описание: получение режима видимости границы ячейки.

Аргументы:

- <строка>, <колонка> — неотрицательные целые числа;
- <граница> — см. метод GetCellGridColor.

Возврат:

- :vlax-true — видима;
- :vlax-false — невидима.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0b938224>
(vla-GetCellGridVisibility obj 2 2 acBottomMask)
;;; :vlax-true
```

7.2.145. Метод *GetCellStyleOverrides*

2005

Объект: **Table**.

Описание: получение номеров наложенных стилей ячейки.

Аргументы:

- <строка>, <колонка> — неотрицательные целые числа.

Возврат: одномерный массив в виде варианта с типом безопасного массива может быть пустым.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0b938224>
(setq temp (vla-GetCellStyleOverrides obj 2 2))
;;; #<variant 8195 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (130 132 131 133 136 138 139 137)
```

7.2.146. Метод *GetCellTextHeight*

Объект: **Table**.

Описание: получение высоты текста ячейки.

Аргументы:

- <строка>, <колонка> — неотрицательные целые числа.

Возврат: положительное вещественное число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0b938224>
(vla-GetCellTextHeight obj 2 2)
;;; 4.5
```

7.2.147. Метод *GetCellTextStyle*

Объект: **Table**.

Описание: получение стиля текста ячейки.

Аргументы:

- <строка>, <колонка> — неотрицательные целые числа.

Возврат: строка.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0b938224>
(vla-GetCellTextStyle obj 2 2)
;;; "Standard"
```

7.2.148. Метод *GetCellType*

Объект: **Table**.

Описание: получение типа содержимого ячейки.

Аргументы:

- <строка>, <колонка> — неотрицательные целые числа.

Возврат:

- `acUnknownCell` (0) — неизвестный тип;
- `acTextCell` (1) — текст;
- `acBlockCell` (2) — блок.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0b938224>
(vla-GetCellType obj 2 2)
;;; 1
```

7.2.149. Метод *GetColor*

Объект: **TableStyle**.

Описание: получение цвета ячеек таблицы.

Аргументы:

- `<тип строк>` — указание типа строк, см. метод `GetAlignment`.

Возврат: объект типа `AcCmColor`.

```
(setq obj (vla-Item (vla-Item (vla-get-Dictionaries actdoc)
    "ACAD_TABLESTYLE") "test"))
;;; #<VLA-OBJECT IAcadTableStyle 0ba380e4>
(vla-GetColor obj acDataRow)
;;; #<VLA-OBJECT IAcadAcCmColor 0ba3d3c0>
```

7.2.150. Метод *GetColumnWidth*

Объект: **Table**.

Описание: получение ширины колонки.

Аргументы:

- `<колонка>` — неотрицательное целое число.

Возврат: положительное вещественное число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0ba39964>
(vla-GetColumnWidth obj 2)
;;; 66.4397
```

7.2.151. Метод *GetConstantAttributes*

Объекты: **BlockRef**, **ExternalReference**, **MInsertBlock**.

Описание: получение постоянных атрибутов.

Аргументы: нет.

Возврат: массив в виде варианта с типом безопасного массива может быть пустым.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadBlockReference 0ba30f74>
(setq temp (vla-GetConstantAttributes obj))
;;; #<variant 8201 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (#<VLA-OBJECT IAcadAttribute 0ba34914> #<VLA-OBJECT IAcadAtt ...)
```

7.2.152. Метод *GetContentColor*

2005

Объект: **Table**.

Описание: получение цвета ячеек заданного типа.

Аргументы:

- <тип строк> — указание типа строк, см. метод GetAlignment.

Возврат: объект типа AcCmColor.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0ba31f74>
(vla-GetContentColor obj acDataRow)
;;; #<VLA-OBJECT IAcadAcCmColor 0ba31550>
```

7.2.153. Метод *GetControlPoint*

Объект: **Spline**.

Описание: получение координат управляющей точки по номеру.

Аргументы:

- <индекс> — положительное целое число, номер точки.

Возврат: вариант с типом безопасного массива.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadSpline 0ba21df4>
(setq temp (vla-GetControlPoint obj 3))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (2308.14 474.063 0.0)
```

7.2.154. Метод *GetCorner*

Объект: **Utility**.

Описание: интерактивное получение точки, с подсветкой прямоугольной рамки, начинающейся из другой точки (с учетом предшествующего вызова метода InitializeUserInput).

Аргументы:

- <точка> — координаты трехмерной точки в виде варианта с типом безопасного массива;
- [<подсказка>] — строка.

Возврат: вариант с типом безопасного массива.

```
(setq obj (vla-get-Utility actdoc))
;;; #<VLA-OBJECT IAcadUtility 0ba20404>
(setq temp (vla-GetCorner obj (vlax-3D-point (getpoint)) "\nКликни: "))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (2746.84 636.576 0.0)
```

7.2.155. Метод *GetCustomByIndex*

Объект: **SummaryInfo**.

Описание: получение пользовательской информации по номеру.

Аргументы:

- <индекс> — положительное целое число;
- <ключ> и <значение> — символы, которым присваиваются строки.

Возврат: nil.

```
(setq obj (vla-get-SummaryInfo actdoc))
;;; #<VLA-OBJECT IAcadSummaryInfo 0ba33064>
(vla-GetCustomByIndex obj 3 'field 'datum)
field
;;; "My Datum "
datum
;;; "Important string"
```

7.2.156. Метод *GetCustomByKey*

Объект: **SummaryInfo**.

Описание: получение пользовательской информации по имени поля (ключа).

Аргументы:

- <имя поля> — строка;
- <значение> — символ, которому присваивается строка.

Возврат: nil.

```
(setq obj (vla-get-SummaryInfo actdoc))
;;; #<VLA-OBJECT IAcadSummaryInfo 0ba33064>
(vla-GetCustomByKey obj "My Datum" 'datum)
datum
;;; "Important string"
```

7.2.157. Метод *GetCustomScale*

Объекты: **Layout**, **PlotConfiguration**.

Описание: получение пользовательского масштаба.

Аргументы:

- <бумага>, <документ> — символы, которым присваиваются положительные вещественные числа.

Возврат: nil.

```
(setq obj (vla-get-ActiveLayout actdoc))
;;; #<VLA-OBJECT IAcadLayout 0ba44594>
(vla-GetCustomScale obj 'paper 'drawing)
paper
;;; 1.0
drawing
;;; 670.2
```

7.2.158. Метод *GetDistance*

Объект: **Utility**.

Описание: интерактивное получение расстояния между двумя точками (с учетом предшествующего вызова метода InitializeUserInput).

Аргументы:

- [<точка>] — координаты трехмерной точки в виде варианта с типом безопасного массива;
- [<подсказка>] — строка.

Примечание

При отсутствии аргумента <точка> для передачи подсказки требуется указать первый аргумент в виде nil.

Возврат: положительное вещественное число.

```
(setq obj (vla-get-Utility actdoc))
;;; #<VLA-OBJECT IAcadUtility 0ba20404>
(vla-GetDistance obj nil "\nКликни пару раз: ")
;;; 145.502
```

7.2.159. Метод *GetDynamicBlockProperties*

Объекты: **BlockRef**, **MInsertBlock**.

Описание: получение изменяемых свойств динамического блока.

Аргументы: нет.

Возврат: массив свойств в виде варианта с типом безопасного массива.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadBlockReference2 0b100d24>
(setq temp (vla-GetDynamicBlockProperties obj))
;;; #<variant 8201 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (#<VLA-OBJECT IAcadDynamicBlockReferenceProperty 0b10bae4> ...)
```

7.2.160. Метод *GetEntity*

Объект: **Utility**.

Описание: интерактивный выбор одиночного примитива.

Аргументы:

- <выбор>, <точка> — символы, которым присваиваются данные;
- [<подсказка>] — строка.

Возврат: nil.

```
(setq obj (vla-get-Utility actdoc))
;;; #<VLA-OBJECT IAcadUtility 0ba20404>
(vla-GetEntity obj 'select 'point "\nВыбери объект: ")
select
;;; #<VLA-OBJECT IAcadArc 0b6f89f4>
(vlax-safearray->list point)
;;; (2621.72 356.952 0.0)
```

7.2.161. Метод *GetExtensionDictionary*

Объекты: **все графические объекты**, **AttributeReference**, **Blocks**, **Dictionaries**, **Dictionary**, **DimStyle**, **DimStyles**, **Group**, **Groups**, **Layer**, **Layers**, **Layout**, **Layouts**,

Linetype, **Linetypes**, **PlotConfiguration**, **PlotConfigurations**, **RegisteredApplication**, **RegisteredApplications**, **TextStyle**, **TextStyles**, **UCS**, **UCSs**, **View**, **Viewport**, **Viewports**, **Views**, **XRecord**.

Описание: получение словаря, связанного с объектом или семейством.

Примечание

Если в момент применения метода объект не имеет словаря, то словарь создается.

Аргументы: нет.

Возврат: VLA-объект.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadMText2 0b6f84c4>
(vla-GetExtensionDictionary obj)
;;; #<VLA-OBJECT IAcadDictionary 0b6fa884>
```

7.2.162. Метод *GetFieldId*

Объект: **Table**.

Описание: получение идентификатора объекта с полем в ячейке.

Аргументы:

- <строка>, <колонка> — неотрицательные целые числа.

Возврат: положительное целое число. Если объекта в ячейке нет — 0.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0b6fa024>
(vla-GetFieldId obj 1 4)
;;; 2130076880
```

7.2.163. Метод *GetFitPoint*

Объект: **Spline**.

Описание: получение координат опорной точки по номеру.

Аргументы:

- <индекс> — положительное целое число, номер точки.

Возврат: вариант с типом безопасного массива.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadSpline 0b6f8f74>
(setq temp (vla-GetFitPoint obj 2))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (2369.62 367.579 0.0)
```

7.2.164. Метод *GetFont*

Объект: **TextStyle**.

Описание: получение параметров текстового стиля.

Аргументы:

- <шрифт> — символ, присваивается строковое значение;
- <полужирный>, <наклонный> — символы, присваивается:

  :vlax-true — да;

  :vlax-false — нет.
- <набор> — символ, присваивается:

  0 — набор Национального института стандартизации США (ANSI);

  1 — по умолчанию;

  2 — символьный набор;

  128 — набор Японских промышленных стандартов (JIS);

  255 — набор производителя аппаратуры (OEM).
- <характер и семейство> — символ, значения складываются:

  характер:

  0 — по умолчанию;

  1 — фиксированный;

  2 — переменный;

  семейство:

  0 — безразличный или неизвестный;

  16 — переменной ширины с засечками;

  32 — переменной ширины рубленый;

  48 — моноширинный;

  64 — рукописный и подобные;

  80 — староанглийский и подобные.

Возврат: nil.

```
(setq obj (vla-get-ActiveTextStyle actdoc))
;;; #<VLA-OBJECT IAcadTextStyle 0b6fbb84>
(vla-GetFont obj 'name 'bold 'italic 'charset 'family)
name
;;; "Book Antiqua"
bold
;;; :vlax-false
```

```
italic
;;; :vlax-false
charset
;;; 0
family
;;; 18
```

7.2.165. Метод *GetFullDrawOrder*

2005

Объект: **SortentsTable**.

Описание: получение порядка сортировки примитивов.

Аргументы:

- <порядок> — символ, присваивается безопасный массив, содержащий указатели на объекты в обратном порядке (последний сверху);
- <учет системной установки> — признак учета системной переменной DRAWORDERCTL:

  `:vlax-true` — учитывать;

  `:vlax-false` — не учитывать.

Возврат: VLA-объект.

```
(setq dict (vla-GetExtensionDictionary (vla-get-ModelSpace actdoc)))
;;; #<VLA-OBJECT IAcadDictionary 0f722cf4>
(setq obj (vla-AddObject dict "ACAD_SORTENTS" "AcDbSortentsTable"))
;;; #<VLA-OBJECT IAcadSortentsTable 0f721ba4>
(vla-GetFullDrawOrder obj 'order :vlax-false)
(vlax-safearray->list order)
;;; (#<VLA-OBJECT IAcadLine 0f74b1c4> #<VLA-OBJECT IAcadLine ...)
```

7.2.166. Метод *GetGridColor*

2005

Объекты: **Table**, **TableStyle**.

Описание: получение цвета сетки ячеек.

Аргументы:

- <тип сетки> — указание типа границ:

  `acInvalidGridLine` (0) — недействительная;

  `acHorzTop` (1) — горизонтальная верхняя;

  `acHorzInside` (2) — горизонтальная внутренняя;

  `acHorzBottom` (4) — горизонтальная нижняя;

  `acVertLeft` (8) — вертикальная левая;

acVertInside (16) — вертикальная внутренняя;

acVertRight (32) — вертикальная правая.

- <тип строк> — указание типа строк, см. метод GetAlignment.

Возврат: объект типа AcCmColor.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0f74d564>
(vla-GetGridColor obj acHorzBottom acDataRow)
;;; #<VLA-OBJECT IAcadAcCmColor 0d70cea0>
```

7.2.167. Метод *GetGridLineWeight*

2005

Объекты: **Table**, **TableStyle**.

Описание: получение веса линий сетки ячеек.

Аргументы:

- <тип сетки> — указание типа границ, см. метод GetGridColor;
- <тип строк> — указание типа строк, см. метод GetAlignment.

Возврат: целое число, см. свойство DimensionLineWeight.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0f74d564>
(vla-GetGridLineWeight obj acHorzBottom acDataRow)
;;; -2
```

7.2.168. Метод *GetGridSpacing*

Объекты: **Viewport**, **PViewport**.

Описание: получение размеров сетки.

Аргументы:

- <ось X>, <ось Y> — символы, которым присваиваются положительные вещественные числа.

Возврат: nil.

```
(setq obj (vla-get-ActivePViewport actdoc))
;;; #<VLA-OBJECT IAcadPViewport2 110eb3a4>
(vla-GetGridSpacing obj 'hor 'vert)
hor
;;; 10.0
vert
;;; 10.0
```

7.2.169. Метод *GetGridVisibility*

2005

Объекты: **Table**, **TableStyle**.

Описание: получение режима отображения линий сетки ячеек.

Аргументы:

- `<тип сетки>` — указание типа границ, см. метод `GetGridColor`;
- `<тип строк>` — указание типа строк, см. метод `GetAlignment`.

Возврат:

- `:vlax-true` — отображаются;
- `:vlax-false` — скрываются.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0f74d564>
(vla-GetGridVisibility obj acHorzBottom acDataRow)
;;; :vlax-true
```

7.2.170. Метод *GetInput*

Объект: **Utility**.

Описание: получение ключевого ввода пользователя.

Аргументы: нет.

Возврат: строка (ключевое слово, введенное пользователем).

```
(setq obj (vla-get-Utility actdoc))
;;; #<VLA-OBJECT IAcadUtility 0bf3bb94>
(vla-InitializeUserInput obj 128 "Any Exit")
(vl-catch-all-apply 'vla-GetPoint (list obj nil "\nPoint [Any Exit]: "))
;;; #<%catch-all-apply-error%>
(vla-GetInput obj)
;;; "Any"
```

7.2.171. Метод *GetInteger*

Объект: **Utility**.

Описание: интерактивный ввод целочисленного значения (с учетом предшествующего вызова метода `InitializeUserInput`).

Аргументы:

- `[<подсказка>]` — строка.

Возврат: целое число.

```
(setq obj (vla-get-Utility actdoc))
;;; #<VLA-OBJECT IAcadUtility 0ba20404>
(vla-GetInteger obj "\nВведи целое число: ")
;;; 45
```

7.2.172. Метод *GetInterfaceObject*

Объект: **Application**.

Описание: получение указателя на объект внешнего приложения.

Аргументы:

- <идентификатор> — строка.

Возврат: указатель (VLA-объект).

```
(setq obj (vlax-get-acad-object))
;;; #<VLA-OBJECT IAcadApplication 00b9d604>
(vla-GetInterfaceObject obj "AutoCAD.AcCmColor.16")
;;; #<VLA-OBJECT IAcadAcCmColor 0bf43e40>
```

7.2.173. Метод *GetInvisibleEdge*

Объект: **3DFace**.

Описание: получение режима видимости граней.

Аргументы:

- <индекс> — целое число от 0 до 3.

Возврат:

- `:vlax-true` — отображаются;
- `:vlax-false` — не отображаются.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcad3DFace 0bf45a84>
(vla-GetInvisibleEdge obj 2)
;;; :vlax-true
```

7.2.174. Метод *GetKeyword*

Объект: **Utility**.

Описание: интерактивный ввод ключевого слова (с учетом предшествующего вызова метода `InitializeUserInput`).

Аргументы:

- [<подсказка>] — строка.

Возврат: строка.

```
(setq obj (vla-get-Utility actdoc))
;;; #<VLA-OBJECT IAcadUtility 0bf3bb94>
(vla-InitializeUserInput obj 128 "Uno Due Tre")
(vla-GetKeyword obj "\nВведи опцию [Uno Due Tre]: ")
;;; "Due"
```

7.2.175. Метод *GetLabelTemplates*

Объект: **DbConnect**.

Описание: получение коллекции (семейства) шаблонов ярлыков.

Аргументы:

- [<документ>] — VLA-объект документа, в контексте которого должен выполняться метод.

Примечание

При отсутствии аргумента коллекция извлекается из текущего документа.

Возврат: созданный VLA-объект.

```
(setq DBConn (vla-GetInterfaceObject (vlax-get-acad-object)
               "CAO.DbConnect.16"))
;;; #<VLA-OBJECT IDbConnect 0c1cc990>
(vlax-invoke DBConn 'GetLabelTemplates)
;;; #<VLA-OBJECT ILabelTemplates 0c1cc7c0>
```

7.2.176. Метод *GetLinks*

Объект: **DbConnect**.

Описание: получение связей.

Аргументы:

- [<шаблон связи>] — VLA-объект или строка, идентифицирующая шаблон;
- [<объекты>] — вариант, содержащий безопасный массив целых чисел — ID графических объектов;
- [<тип связей>] — см. свойство LinkType;
- [<документ>] — VLA-объект документа, в контексте которого должен выполняться метод.

Примечание

При отсутствии аргумента связи извлекаются из текущего документа.

Возврат: созданный VLA-объект.

```
(setq DBConn (vla-GetInterfaceObject (vlax-get-acad-object)
               "CAO.DbConnect.16"))
;;; #<VLA-OBJECT IDbConnect 04de13f0>
(setq DBLinksT (vlax-invoke DBConn 'GetLinkTemplates))
;;; #<VLA-OBJECT ILinkTemplates 04df12d0>
(setq DBLinkT (vla-Item DBLinksT "Link2"))
;;; #<VLA-OBJECT ILinkTemplate 04dff0d0>
(vlax-invoke-method DBConn 'GetLinks DBLinkT (vlax-make-variant
  (vlax-safearray-fill (vlax-make-safearray vlax-vbLong '(0 . 0))
    (list (vla-get-ObjectID (vlax-ename->vla-object (car (entsel)))))))
  2 actdoc)
;;; #<VLA-OBJECT ILinks 0c21b3d0>
```

7.2.177. Метод *GetLinksByKey*

Объект: **DbConnect**.

Описание: получение связей по значениям ключей.

Аргументы:

- <шаблон связи> — VLA-объект или строка, идентифицирующая шаблон;
- <значения ключей> — VLA-объект, KeyValues;

Примечание

Порядок создания, количество и содержимое ключей должны соответствовать используемому шаблону связи.

- [<тип связей>] — см. свойство LinkType;
- [<документ>] — VLA-объект документа, в контексте которого должен выполняться метод.

Примечание

При отсутствии аргумента связи извлекаются из текущего документа.

Возврат: созданный VLA-объект.

```
(setq DBConn (vla-GetInterfaceObject (vlax-get-acad-object)
               "CAO.DbConnect.16"))
;;; #<VLA-OBJECT IDbConnect 0bbc3750>
(setq DBLinksT (vlax-invoke DBConn 'GetLinkTemplates))
;;; #<VLA-OBJECT ILinkTemplates 0bbc3870>
(setq DBLinkT (vla-Item DBLinksT "Link2"))
;;; #<VLA-OBJECT ILinkTemplate 0bbc54c0>
(setq KeyVals (vla-GetInterfaceObject (vlax-get-acad-object)
    "CAO.KeyValues.16"))
;;; #<VLA-OBJECT IKeyValues 0bbc5b30>
```

```
(vlax-invoke KeyVals 'Add "MANUFACTURER" "HEWLETT PACKARD")
;;; #<VLA-OBJECT IKeyValue 0bbc5250>
(vlax-invoke KeyVals 'Add "Equipment_Description" "LASERJET 2")
;;; #<VLA-OBJECT IKeyValue 0c1f6570>
(vlax-invoke-method DBConn 'GetLinksByKey DBLinkT KeyVals 2 actdoc)
;;; #<VLA-OBJECT ILinks 0bbc57e0>
```

7.2.178. Метод *GetLinksByLabelTemplate*

Объект: **DbConnect**.

Описание: получение связей по шаблону ярлыка.

Аргументы:

- <шаблон ярлыка> — VLA-объект или строка, идентифицирующая шаблон;
- [<тип связей>] — см. свойство LinkType;
- [<документ>] — VLA-объект документа, в контексте которого должен выполняться метод.

Примечание

При отсутствии аргумента связи извлекаются из текущего документа.

Возврат: созданный VLA-объект.

```
(setq DBConn (vla-GetInterfaceObject (vlax-get-acad-object)
               "CAO.DbConnect.16"))
;;; #<VLA-OBJECT IDbConnect 0bbc3750>
(setq DBLablsT (vlax-invoke DBConn 'GetLabelTemplates))
;;; #<VLA-OBJECT ILabelTemplates 0bbc3560>
(setq DBLablT (vla-Item DBLablsT "Label4"))
;;; #<VLA-OBJECT ILabelTemplate 0bbbfa60>
(vlax-invoke-method DBConn 'GetLinksByLabelTemplate DBLablT 2 actdoc)
;;; #<VLA-OBJECT ILinks 0bbbff70>
```

7.2.179. Метод *GetLinksBySQLCond*

Объект: **DbConnect**.

Описание: получение связей по SQL-запросу.

Аргументы:

- <шаблон связи> — VLA-объект или строка, идентифицирующая шаблон;
- <выражение> — строка, SQL-запрос;
- [<тип связей>] — см. свойство LinkType;
- [<документ>] — VLA-объект документа, в контексте которого должен выполняться метод.

Примечание

При отсутствии аргумента связи извлекаются из текущего документа.

Возврат: созданный VLA-объект.

```
(setq DBConn (vla-GetInterfaceObject (vlax-get-acad-object)
               "CAO.DbConnect.16"))
;;; #<VLA-OBJECT IDbConnect 0bbc3750>
(setq DBLinksT (vlax-invoke DBConn 'GetLinkTemplates))
;;; #<VLA-OBJECT ILinkTemplates 0bbc3870>
(setq DBLinkT (vla-Item DBLinksT "Link2"))
;;; #<VLA-OBJECT ILinkTemplate 0bbc54c0>
(vlax-invoke-method DBConn 'GetLinksBySQLCond DBLinkT
    "Manufacturer = 'Mitsubishi'" 2 actdoc)
;;; #<VLA-OBJECT ILinks 0bbb2f80>
```

7.2.180. Метод *GetLinkTemplates*

Объект: **DbConnect**.

Описание: получение шаблонов связей.

Аргументы:

- [<документ>] — VLA-объект документа, в контексте которого должен выполняться метод.

Примечание

При отсутствии аргумента шаблоны извлекаются из текущего документа.

Возврат: VLA-объект.

```
(setq DBConn (vla-GetInterfaceObject (vlax-get-acad-object)
               "CAO.DbConnect.16"))
;;; #<VLA-OBJECT IDbConnect 0f8184d0>
(vlax-invoke DBConn 'GetLinkTemplates)
;;; #<VLA-OBJECT ILinkTemplates 0f81cb20>
```

7.2.181. Метод *GetLocaleMediaName*

Объекты: **Layout**, **PlotConfiguration**.

Описание: получение локализованного имени формата листа из канонического.

Аргументы:

- <имя> — строка, каноническое имя.

Возврат: строка.

```
(setq obj (vla-get-ActiveLayout actdoc))
;;; #<VLA-OBJECT IAcadLayout 0bf46844>
(vla-GetLocaleMediaName obj "Envelope B5")
;;; "Конверт B5"
```

7.2.182. Метод *GetLoopAt*

Объект: **Hatch**.

Описание: получение границы штриховки по номеру.

Аргументы:

- <индекс> — неотрицательное целое число, номер границы;
- <граница> — символ, которому присваивается указатель на безопасный массив объектов, составляющих границу.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadHatch 0bf3b9f4>
(vla-GetLoopAt obj 0 'loop)
(vlax-safearray->list loop)
;;; (#<VLA-OBJECT IAcadCircle 0bf3ca84> #<VLA-OBJECT IAcadCirc ...)
```

7.2.183. Метод *GetMinimumColumnWidth* 2005

Объект: **Table**.

Описание: получение минимальной ширины колонки.

Аргументы:

- <номер колонки> — неотрицательное целое число.

Возврат: положительное вещественное число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0bf38b44>
(vla-GetMinimumColumnWidth obj 0)
;;; 7.5
```

7.2.184. Метод *GetMinimumRowHeight* 2005

Объект: **Table**.

Описание: получение минимальной высоты строки.

Аргументы:

- <номер строки> — неотрицательное целое число.

Возврат: положительное вещественное число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0bf38b44>
(vla-GetMinimumRowHeight obj 2)
;;; 9.0
```

7.2.185. Метод *GetName*

Объект: **Dictionary**.

Описание: получение имени вложенного объекта.

Аргументы:

- <объект> — VLA-объект.

Возврат: строка.

```
(setq obj (vla-Item (vla-get-Dictionaries actdoc) "MyTestDict"))
;;; #<VLA-OBJECT IAcadDictionary 0bf3c4f4>
(vla-GetName obj (vla-Item obj 0))
;;; "MyTestRec"
```

7.2.186. Метод *GetObject*

Объект: **Dictionary**.

Описание: получение вложенного объекта.

Аргументы:

- <имя> — строка, имя вложенного объекта.

Возврат: VLA-объект.

```
(setq obj (vla-Item (vla-get-Dictionaries actdoc) "MyTestDict"))
;;; #<VLA-OBJECT IAcadDictionary 0bf3c4f4>
(vla-GetObject obj "MyTestRec")
;;; #<VLA-OBJECT IAcadXRecord 0bf39d04>
```

7.2.187. Метод *GetOrientation*

Объект: **Utility**.

Описание: интерактивное получение угла от оси X в радианах (с учетом предшествующего вызова метода InitializeUserInput).

Аргументы:

- [<точка>] — координаты трехмерной точки в виде варианта с типом безопасного массива;
- [<подсказка>] — строка.

Примечание

При отсутствии аргумента <точка> для передачи подсказки требуется указать первый аргумент в виде nil.

Возврат: положительное вещественное число.

```
(setq obj (vla-get-Utility actdoc))
;;; #<VLA-OBJECT IAcadUtility 0bf3bb94>
(vla-GetOrientation obj nil "\nКликни пару раз: ")
;;; 0.445024
```

7.2.188. Метод *GetPaperMargins*

Объекты: **Layout**, **PlotConfiguration**.

Описание: получение полей печатного листа.

Аргументы:

- <первый>, <второй> — символы, которым присваиваются безопасные массивы, содержащие положительные вещественные числа — значения полей, в следующем порядке:

  первый — левое, нижнее;

  второй — верхнее, правое.

Возврат: nil.

```
(setq obj (vla-get-ActiveLayout actdoc))
;;; #<VLA-OBJECT IAcadLayout 0bf39844>
(vla-GetPaperMargins obj 'first 'second)
(vlax-safearray->list first)
;;; (5.79375 17.7938)
(vlax-safearray->list second)
;;; (5.79376 17.7938)
```

7.2.189. Метод *GetPaperSize*

Объекты: **Layout**, **PlotConfiguration**.

Описание: получение размера печатного листа.

Аргументы:

- <ширина>, <высота> — символы, которым присваиваются значения.

Возврат: nil.

```
(setq obj (vla-get-ActiveLayout actdoc))
;;; #<VLA-OBJECT IAcadLayout 0bf39844>
(vla-GetPaperSize obj 'width 'height)
```

```
width
;;; 594.0
height
;;; 420.0
```

7.2.190. Метод *GetPlotDeviceNames*

Объекты: **Layout**, **PlotConfiguration**.

Описание: получение имен печатных устройств.

Аргументы:

- <имена> — символ, которому присваивается вариант с типом безопасного массива строк.

Возврат: массив строк в виде варианта.

```
(setq obj (vla-get-ActiveLayout actdoc))
;;; #<VLA-OBJECT IAcadLayout 0bf39844>
(setq temp (vla-GetPlotDeviceNames obj))
;;; #<variant 8200 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; ("None" "Microsoft Office Document Image Writer" "Adobe ...)
```

7.2.191. Метод *GetPlotStyleTableNames*

Объекты: **Layout**, **PlotConfiguration**.

Описание: получение имен таблиц стилей печати.

Аргументы:

- <имена> — символ, которому присваивается вариант с типом безопасного массива строк.

Возврат: массив строк в виде варианта.

```
(setq obj (vla-get-ActiveLayout actdoc))
;;; #<VLA-OBJECT IAcadLayout 0bf39844>
(setq temp (vla-GetPlotStyleTableNames obj))
;;; #<variant 8200 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; ("acad.stb" "AIA Color LWT by Object.stb" "AIA LWT by ...)
```

7.2.192. Метод *GetPoint*

Объект: **Utility**.

Описание: интерактивное получение координат точки (с учетом предшествующего вызова метода `InitializeUserInput`).

Аргументы:

- [<точка>] — координаты трехмерной точки в виде варианта с типом безопасного массива;
- [<подсказка>] — строка.

Примечание

При отсутствии аргумента <точка> для передачи подсказки требуется указать первый аргумент в виде nil.

Возврат: координаты точки в виде варианта с типом безопасного массива.

```
(setq obj (vla-get-Utility actdoc))
;;; #<VLA-OBJECT IAcadUtility 0bf3bb94>
(setq temp (vla-GetPoint obj nil "\nУкажи точку: "))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (2672.93 396.983 0.0)
```

7.2.193. Метод *GetProjectFilePath*

Объект: **PreferencesFiles**.

Описание: получение пути к файлам проекта.

Аргументы:

- <проект> — строковый, имя проекта.

Возврат: строка.

```
(setq obj (vla-get-Files
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesFiles2 0b8361e0>
(vla-GetProjectFilePath obj (getvar "PROJECTNAME"))
;;; "D:\\Tutor"
```

7.2.194. Метод *GetQueries*

Объект: **DbConnect**.

Описание: получение сохраненных SQL-запросов.

Аргументы:

- [<документ>] — VLA-объект документа, в контексте которого должен выполняться метод.

Примечание

При отсутствии аргумента запросы извлекаются из текущего документа.

Возврат: VLA-объект.

```
(setq DBConn (vla-GetInterfaceObject (vlax-get-acad-object)
               "CAO.DbConnect.16"))
;;; #<VLA-OBJECT IDbConnect 0f8184d0>
(vlax-invoke DBConn 'GetQueries)
;;; #<VLA-OBJECT IQueries 0f81c930>
```

7.2.195. Метод *GetReal*

Объект: **Utility**.

Описание: интерактивное получение вещественного числа (с учетом предшествующего вызова метода `InitializeUserInput`).

Аргументы:

- `[<подсказка>]` — строка.

Возврат: положительное вещественное число.

```
(setq obj (vla-get-Utility actdoc))
;;; #<VLA-OBJECT IAcadUtility 0bf3bb94>
(vla-GetReal obj "\nВведи число: ")
;;; 4654.0
```

7.2.196. Метод *GetRelativeDrawOrder* 2005

Объект: **SortentsTable**.

Описание: получение относительного порядка сортировки примитивов.

Аргументы:

- `<объекты>` — безопасный массив, содержащий проверяемые объекты;
- `<учет системной установки>` — признак учета системной переменной DRAWORDERCTL:

  `:vlax-true` — учитывать;

  `:vlax-false` — не учитывать.

Возврат: `nil`.

```
(setq dict (vla-GetExtensionDictionary (vla-get-ModelSpace actdoc)))
;;; #<VLA-OBJECT IAcadDictionary 0b833354>
(setq obj (vla-GetObject dict "ACAD_SORTENTS"))
;;; #<VLA-OBJECT IAcadSortentsTable 0b837c54>
(setq temp (mapcar '(lambda (a) (vlax-ename->vla-object (car a)))
    (list (entsel) (entsel))))
;;; (#<VLA-OBJECT IAcadLWPolyline 08ee8fa4> #<VLA-OBJECT IAcadCircle...)
```

```
(setq temp (vlax-safearray-fill
 (vlax-make-safearray vlax-vbObject '(0 . 1)) temp))
;;; #<safearray...>
(vla-GetRelativeDrawOrder obj temp :vlax-false)
;;; error: Automation Error. Invalid object array
```

7.2.197. Метод *GetRemoteFile*

Объект: **Utility**.

Описание: копирование файла из внешнего источника.

Аргументы:

- <ссылка> — строка, URL;
- <файл> — символ, которому присваивается строка, имя копии файла;
- <не использовать кэш> — признак игнорирования кэша:

  :vlax-true — игнорировать;

  :vlax-false — учитывать.

Возврат: nil.

```
(setq obj (vla-get-Utility actdoc))
;;; #<VLA-OBJECT IAcadUtility 0b819b84>
(vla-GetRemoteFile obj "http://www.uniip.ru/l...t.zip" 'temp :vlax-false)
temp
;;; "C:\\Doc ... Content.IE5\\RT431B9Z\\AddAtt(0.8)[1].8).zip"
```

7.2.198. Метод *GetRowHeight*

Объект: **Table**.

Описание: получение высоты строки.

Аргументы:

- <номер строки> — неотрицательное целое число.

Возврат: положительное вещественное число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0b8030b4>
(vla-GetRowHeight obj 2)
;;; 9.0
```

7.2.199. Метод *GetRowType*

Объект: **Table**.

Описание: получение типа строки.

Аргументы:

- <номер строки> — неотрицательное целое число.

Возврат: целое число, интерпретацию см. метод GetAlignment.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0b8030b4>
(vla-GetRowType obj 2)
;;; 1
```

7.2.200. Метод *GetSnapSpacing*

Объекты: **PViewport**, **Viewport**.

Описание: получение параметров режима SNAP (ШАГ).

Аргументы:

- <ось X>, <ось Y> — символы, которым присваиваются значения.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadPViewport2 0b84cc74>
(vla-GetSnapSpacing obj 'stepx 'stepy)
stepx
;;; 10.0
stepy
;;; 10.0
```

7.2.201. Метод *GetString*

Объект: **Utility**.

Описание: интерактивное получение строки (с учетом предшествующего вызова метода InitializeUserInput).

Аргументы:

- <разрешение пробелов> — признак разрешения пробелов в строке:

  :vlax-true — разрешить;

  :vlax-false — запретить.

- [<подсказка>] — строка.

Возврат: строка.

```
(setq obj (vla-get-Utility actdoc))
;;; #<VLA-OBJECT IAcadUtility 0b819b84>
(vla-GetString obj :vlax-false "\nВведи строку: ")
;;; "Anystring"
```

7.2.202. Метод *GetSubEntity*

Объект: **Utility**.

Описание: интерактивное получение примитива или подпримитива (при выборе комплексного примитива).

Аргументы:

- <объект>, <точка>, <матрица трансформации>, <список идентификаторов вложенности> — символы, которым присваиваются данные;
- [<подсказка>] — строка.

Возврат: nil.

```
(setq obj (vla-get-Utility actdoc))
;;; #<VLA-OBJECT IAcadUtility 0995e8d4>
(vla-GetSubEntity obj 'ent 'point 'mtx 'context "\nВыбери объект: ")
ent
;;; #<VLA-OBJECT IAcadAttribute 0b9d79d4>
(vlax-safearray->list point)
;;; (2532.17 559.947 0.0)
(vlax-safearray->list mtx)
;;; ((1.0 0.0 0.0 0.0) (0.0 1.0 0.0 0.0) (0.0 0.0 1.0 0.0)
;;; (2517.77 618.767 0.0 1.0))
(setq temp (vlax-safearray->list context))
;;; (2130062896)
(vla-ObjectIDToObject actdoc (car temp))
;;; #<VLA-OBJECT IAcadBlockReference 0b9d60a4>
```

7.2.203. Метод *GetSubSelection*

Объект: **Table**.

Описание: получение индексов выборки.

Примечание

См. также метод SetSubSelection.

Аргументы:

- <строка мин.>, <строка макс.>, <колонка мин.>, <колонка макс.> — символы, которым присваиваются индексы.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0b9d9624>
(vla-GetSubSelection obj 'rowmin 'rowmax 'colmin 'colmax)
```

```
rowmin
;;; 0
rowmax
;;; 2
colmin
;;; 1
colmax
;;; 3
```

7.2.204. Метод *GetText*

2005

Объект: **Table**.

Описание: получение текста из ячейки.

Аргументы:

- <строка>, <колонка> — неотрицательные целые числа.

Возврат: строка.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0b6fa024>
(vla-GetText obj 1 2)
;;; "55"
```

7.2.205. Метод *GetTextHeight*

2005

Объекты: **Table**, **TableStyle**.

Описание: получение высоты содержимого ячеек таблицы.

Аргументы:

- <тип строк> — указание типа строк:

  acUnknownRow (0) — не определен;

  acDataRow (1) — ячейки с данными;

  acTitleRow (2) — заголовок таблицы;

  acHeaderRow (4) — заголовки колонок.

Возврат: положительное вещественное число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 11c8ce04>
(vla-GetTextHeight obj acDataRow)
;;; 4.5
```

7.2.206. Метод *GetTextRotation*

2005

Объект: **Table**.

Описание: получение ориентации текста в ячейке.

Аргументы:

- <строка>, <колонка> — неотрицательные целые числа.

Возврат: целое число, интерпретируемое как:

- acDegreesUnknown (–1) — неизвестно;
- acDegrees000 (0) — 0°;
- acDegrees090 (1) — 90°;
- acDegrees180 (2) — 180°;
- acDegrees270 (3) — 270°.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0b6fa024>
(vla-GetTextRotation obj 1 2)
;;; 1
```

7.2.207. Метод *GetTextStyle*

2005

Объекты: **Table**, **TableStyle**.

Описание: получение текстового стиля ячеек таблицы.

Аргументы:

- <тип строк> — указание типа строк:

  acUnknownRow (0) — не определен;

  acDataRow (1) — ячейки данных;

  acTitleRow (2) — заголовок таблицы;

  acHeaderRow (4) — заголовки колонок.

Возврат: строка.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 11c8ce04>
(vla-GetTextStyle obj acDataRow)
;;; "Standard"
```

7.2.208. Метод *GetUCSMatrix*

Объект: **UCS**.

Описание: получение матрицы для преобразований в указанную ПСК.

Примечание

Матрица используется в методе `TransformBy`.

Аргументы: нет.

Возврат: вариант с типом безопасного массива, содержащий матрицу 4×4.

```
(setq obj (vla-Item (vla-get-UserCoordinateSystems actdoc) "Test"))
;;; #<VLA-OBJECT IAcadUCS 0bf9ea34>
(setq temp (vla-GetUCSMatrix obj))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; ((0.483843 0.812167 -0.326007 0.0) (0.860165 -0.372678 0.348176 0.0)
;;;  (0.161281 -0.448883 -0.878916 0.0) (0.0 0.0 0.0 1.0))
```

7.2.209. Метод *GetVariable*

Объект: **Document**.

Описание: получение значения системной переменной.

Аргументы:

- <имя> — строка, имя переменной.

Возврат: значение переменной в виде варианта.

```
(setq temp (vla-GetVariable actdoc "cmdecho"))
;;; #<variant 2 1>
(vlax-variant-value temp)
;;; 1
```

7.2.210. Метод *GetWeight*

Объект: **Spline**.

Описание: получение веса управляющей точки по номеру. Метод неприменим к рациональным сплайнам.

Аргументы:

- <индекс> — положительное целое число, номер точки.

Возврат: вещественное число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadSpline 0bfa1f44>
(vla-GetWeight obj 3)
;;; -1.0
```

7.2.211. Метод *GetWidth*

Объекты: **LightweightPolyline**, **Polyline**.

Описание: получение начальной и конечной ширины сегмента.

Аргументы:

- <индекс> — неотрицательное целое, номер сегмента (с нуля);
- <начало>, <конец> — символы, которым присваиваются неотрицательные вещественные числа.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadLWPolyline 0bfa7fa4>
(vla-GetWidth obj 3 'StartWidth 'EndWidth)
StartWidth
;;; 5.0
EndWidth
;;; 25.0
```

7.2.212. Метод *GetWindowToPlot*

Объекты: **Layout**, **PlotConfiguration**.

Описание: получение координат окна (рамки), которое будет выводиться на принтер.

Аргументы:

- <левый нижний>, <правый верхний> — символы, которым присваиваются координаты углов в виде безопасного массива.

Возврат: nil.

```
(setq obj (vla-get-ActiveLayout actdoc))
;;; #<VLA-OBJECT IAcadLayout 0bfa4ae4>
(vla-GetWindowToPlot obj 'LowerLeft 'UpperRight)
(vlax-safearray->list LowerLeft)
;;; (40.616 8.85766)
(vlax-safearray->list UpperRight)
;;; (411.882 261.112)
```

7.2.213. Метод *GetXData*

Объекты: **все графические объекты**, **AttributeReference**, **Block**, **Dictionary**, **DimStyle**, **Group**, **Layer**, **Linetype**, **PlotConfigurations**, **RegisteredApplication**, **TextStyle**, **UCS**, **View**, **Viewport**, **XRecord**.

Описание: получение расширенных данных примитива, созданных приложением ObjectARX, VBA или Visual LISP.

Примечание

См. также метод `SetXData`.

Аргументы:

- `<приложение>` — строка, имя приложения;
- `<тип>`, `<значения>` — символы, которым присваиваются значения в виде варианта с типом безопасного массива.

Возврат: `nil`.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadLine 0b858eb4>
(vla-GetXData obj "TestApp" 'XDataType 'XDataValue)
(vlax-safearray->list XDataType)
;;; (1001 1000 1040)
(setq temp (vlax-safearray->list XDataValue))
;;; (#<variant 8 TestApp> #<variant 8 Description> #<variant 5 12.34>)
(mapcar 'vlax-variant-value temp)
;;; ("TestApp" "Description" 12.34)
```

7.2.214. Метод *GetXRecordData*

Объект: **XRecord**.

Описание: получение данных из записи словаря.

Примечание

См. также метод `SetXRecordData`.

Аргументы:

- `<тип>`, `<значения>` — символы, которым присваиваются значения в виде варианта с типом безопасного массива.

Возврат: `nil`.

```
(setq obj (vla-Item (vla-Item (vla-get-Dictionaries actdoc) "MyTestDict")
    "MyTestXRec"))
;;; #<VLA-OBJECT IAcadXRecord 0b92df74>
(vla-GetXRecordData obj 'XRecDataType 'XRecDataValue)
(vlax-safearray->list XRecDataType)
;;; (2 140)
(setq temp (vlax-safearray->list XRecDataValue))
;;; (#<variant 8 Description> #<variant 5 12.34>)
```

```
(mapcar 'vlax-variant-value temp)
;;; ("Description" 12.34)
```

7.2.215. Метод *HandleToObject*

Объект: **Document**.

Описание: получение объекта по метке.

Аргументы:

- <метка> — строка.

Возврат: VLA-объект.

```
(vla-HandleToObject actdoc "84")
;;; #<VLA-OBJECT IAcadBlockReference 0b928ac4>
```

7.2.216. Метод *Hide*

Объекты: **LabelTemplate**, **Label**.

Описание: установка режима невидимости. Метод, обратный методу Show.

Аргументы: нет. Возврат: nil.

```
(setq DBConn (vla-GetInterfaceObject (vlax-get-acad-object)
               "CAO.DbConnect.16"))
;;; #<VLA-OBJECT IDbConnect 0f8184d0>
(setq DBLinksT (vlax-invoke DBConn 'GetLinkTemplates))
;;; #<VLA-OBJECT ILinkTemplates 0f83c510>
(setq DBLinkT (vla-Item DBLinksT "Link2"))
;;; #<VLA-OBJECT ILinkTemplate 0f83c010>
(setq DBLinks (vlax-invoke-method DBConn 'GetLinks DBLinkT
 (vlax-make-variant (vlax-safearray-fill (vlax-make-safearray
    vlax-vbLong '(0 . 0)) (list (vla-get-ObjectID
      (vlax-ename->vla-object (car (entsel))))))) 2 actdoc))
;;; #<VLA-OBJECT ILinks 0f83cf40>
(setq DBLink (vla-Item DBLinks 0))
;;; #<VLA-OBJECT ILink 0f83ccf0>
(setq DBLabl (vlax-get-property DBLink 'Label))
;;; #<VLA-OBJECT ILabel 0f838820>
(vlax-invoke-method DBLabl 'Hide)
```

7.2.217. Метод *Highlight*

Объекты: **все графические объекты**, **AttributeReference**, **Group**, **SelectionSet**.

Описание: установка режима подсветки.

Аргументы:

- <режим>:

  :vlax-true — подсветить;

  :vlax-false — снять подсветку.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadCircle 0b92fc64>
(vla-Highlight obj :vlax-true)
```

7.2.218. Метод *HitTest*

2005

Объект: **Table**.

Описание: поиск ячейки, которой принадлежит или на которую проектируется точка.

Аргументы:

- <тестовая точка>, <направление> — координаты трехмерных точек в виде варианта с типом безопасного массива;
- <строка>, <колонка> — символы, которым присваиваются неотрицательные целые числа.

Возврат:

- :vlax-true — найдена;
- :vlax-false — неудача.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0b929944>
(vla-HitTest obj (vlax-3D-point (getpoint))
    (vlax-3D-point '(0.0 0.0 1.0)) 'row 'col)
;;; :vlax-false
row
;;; 4
col
;;; 2
```

7.2.219. Метод *Import*

Объекты: **Document**, **LayerStateManager**.

Описание: импорт рисунка или состояния слоев из файла.

Аргументы (различные случаи):

- **Document**
  - <файл> — строка, имя файла;
  - <точка вставки> — координаты трехмерной точки в виде варианта с типом безопасного массива;
  - <масштаб> — положительное вещественное число.
- **LayerStateManager**
  - <файл> — строка, имя файла.

Возврат: созданный VLA-объект или `nil`.

```
(vla-Import actdoc "D:/Temp/test.wmf" (vlax-3D-point (getpoint)) 0.75)
;;; #<VLA-OBJECT IAcadBlockReference 127f6284>
;;; Импортирован рисунок и вставлен как блок.
(setq obj (vla-GetInterfaceObject (vlax-get-acad-object)
    "AutoCAD.AcadLayerStateManager.16"))
;;; #<VLA-OBJECT IAcadLayerStateManager 127f86c4>
(vla-SetDatabase obj (vla-get-Database (vla-get-ModelSpace actdoc)))
;;; Инициализация объекта.
(vla-Import obj "D:/Temp/ExState.las")
```

7.2.220. Метод *ImportProfile*

Объект: **PreferencesProfiles**.

Описание: импорт профиля из внешнего файла.

Аргументы:

- <профиль> — строка, новое имя профиля;
- <файл> — строка, имя файла с расширением arg;
- <включение пути>:

  `:vlax-true` — включить путь в реестр;

  `:vlax-false` — не включать путь.

Возврат: `nil`.

```
(setq obj (vla-get-Profiles
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesProfiles 0bfa454c>
(vla-ImportProfile obj "NewTest" "d:/temp/test.arg" :vlax-false)
```

7.2.221. Метод *IndexOf*

Объект: **FileDependencies**.

Описание: получение номера в таблице зависимостей.

Аргументы:

- <признак> — строка, описание приложения или особенностей;
- <файл> — строка, полное имя файла.

Возврат: целое число, счетчик в таблице зависимостей.

```
(setq obj (vla-get-FileDependencies actdoc))
;;; #<VLA-OBJECT IAcadFileDependencies 0bfa7014>
(vla-IndexOf obj "Acad:Image" "C:\\Doc ... es\\Window.jpg")
;;; 3
```

7.2.222. Метод *InitializeUserInput*

Объект: **Utility**.

Описание: инициализация пользовательского ввода объектным методом, аналогично LISP-функции `initget`. Влияет на последующую работу методов `GetAngle`, `GetCorner`, `GetDistance`, `GetInteger`, `GetKeyword`, `GetOrientation`, `GetPoint`, `GetReal`, `GetString`.

Аргументы:

- <битовый код> — сумма битов, определяющая режим ввода:

  1 — запрет пустого ввода;

  2 — запрет нулевого значения;

  4 — запрет отрицательного значения;

  8 — не учитывать лимиты рисунка;

  16 — зарезервировано для использования в будущем;

  32 — использовать пунктирную резиновую линию;

  64 — не учитывать координату *Z*;

  128 — разрешение произвольного типа ввода.
- [<ключи>] — строка опций, разделенных пробелами.

Возврат: `nil`.

```
(setq obj (vla-get-Utility actdoc))
;;; #<VLA-OBJECT IAcadUtility 0b821dd4>
(vla-InitializeUserInput obj (+ 2 128) "New Old Some")
```

```
(setq temp (vl-catch-all-apply 'vla-GetInteger
    (list obj "\nEnter integer [New/Old/Some] <Exit>: ")))
;;; #<%catch-all-apply-error%>
(if (vl-catch-all-error-p temp) (vla-GetInput obj))
;;; "Old"
```

7.2.223. Метод *InsertBlock*

Объекты: **ModelSpace**, **PaperSpace**, **Block**.

Описание: вставка блока, в том числе внешнего файла, как блока.

Аргументы:

- <точка вставки> — координаты трехмерной точки в виде варианта с типом безопасного массива;
- <имя> — строка, имя блока или файла (определяется по наличию расширения);
- <масштаб X>, <масштаб Y>, <масштаб Z> — вещественные числа;
- <поворот> — угол в радианах, вещественное число;
- [<пароль>] — строка (если вставляемый файл защищен паролем).

Возврат: созданный VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 0b8fba24>
(vla-InsertBlock obj (vlax-3D-point (getpoint)) "test" 0.8 1 1 (/ pi 6))
;;; #<VLA-OBJECT IAcadBlockReference 0b8fa5c4>
```

7.2.224. Метод *InsertColumns*

Объект: **Table**.

Описание: вставка колонок.

Аргументы:

- <колонка> — неотрицательное целое число, номер начала вставки;
- <ширина> — положительное вещественное число;
- <количество> — положительное целое число, количество вставляемых колонок.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0b6fa024>
(vla-InsertColumns obj 2 20 2)
```

7.2.225. Метод *InsertInMenuBar*

Объект: **PopupMenu**.

Описание: добавление меню в строку меню.

Аргументы:

- <индекс> — неотрицательное целое число или строка.

Примечание

Если добавляемое меню уже есть в строке меню, то применение данного метода приводит к ошибке.

Возврат: nil.

```
(setq obj (vla-Item (vla-get-Menus (vla-Item (vla-get-MenuGroups
    (vlax-get-acad-object)) "dbConnect")) 1))
;;; #<VLA-OBJECT IAcadPopupMenu 0113b0f8>
(vla-InsertInMenuBar obj "&Tools")
(vla-RemoveFromMenuBar obj)
;;; Удалили меню из строки меню, чтобы не замусоривать ее.
```

7.2.226. Метод *InsertLoopAt*

Объект: **Hatch**.

Описание: добавление островка к штриховке в заданном порядке.

Аргументы:

- <индекс> — неотрицательное целое число или строка;
- <тип островка>:

  acHatchLoopTypeDefault (0) — по умолчанию;

  acHatchLoopTypeExternal (1) — внешний;

  acHatchLoopTypePolyline (2) — полилиния;

  acHatchLoopTypeDerived (4) — производный;

  acHatchLoopTypeTextbox (8) — текстовый блок.
- <граница> — массив объектов, образующих один замкнутый контур. Допустимы следующие объекты: Line, Polyline, Circle, Ellipse, Spline, Region.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadHatch 0b92c4e4>
(setq temp (list (vlax-ename->vla-object (car (entsel)))))
;;; (#<VLA-OBJECT IAcadLWPolyline 0b911d94>)
```

```
(setq temp (vlax-safearray-fill (vlax-make-safearray vlax-vbObject
    '(0 . 0)) temp))
;;; #<safearray...>
(vla-InsertLoopAt obj 1 acHatchLoopTypeDefault temp)
```

7.2.227. Метод *InsertMenuInMenuBar*

Объект: **PopupMenus**.

Описание: добавление меню в строку меню по имени.

Аргументы:

- <имя меню> — строка;
- <индекс> — неотрицательное целое число или строка.

Примечание

Если добавляемое меню уже есть в строке меню, то применение данного метода приводит к ошибке.

Возврат: nil.

```
(setq obj (vla-get-Menus (vla-Item (vla-get-MenuGroups
    (vlax-get-acad-object)) "dbConnect")))
;;; #<VLA-OBJECT IAcadPopupMenus 0b9283d4>
;;; для 2005:
(vla-InsertMenuInMenuBar obj "D&ata View" "&Tools")
(vla-RemoveFromMenuBar (vla-Item obj "D&ata View"))
;;; Удалили меню из строки меню, чтобы не замусоривать ее.
;;; для 2006:
(vla-InsertMenuInMenuBar obj "Data View" "&Tools")
(vla-RemoveFromMenuBar (vla-Item obj "Data View"))
;;; Удалили меню из строки меню, чтобы не замусоривать ее.
```

7.2.228. Метод *InsertRows*

Объект: **Table**.

Описание: вставка строк.

Аргументы:

- <строка> — неотрицательное целое число, номер начала вставки;
- <высота> — положительное вещественное число;
- <количество> — положительное целое число, количество вставляемых строк.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0b6fa024>
(vla-InsertRows obj 2 8 2)
```

7.2.229. Метод *Intersect*

Объект: **Links**.

Описание: получение связей, общих для двух коллекций.

Аргументы:

- <связи> — VLA-объект, коллекция связей.

Возврат: VLA-объект.

```
(setq DBConn (vla-GetInterfaceObject (vlax-get-acad-object)
               "CAO.DbConnect.16"))
;;; #<VLA-OBJECT IDbConnect 0f8184d0>
(setq DBLinksT (vlax-invoke DBConn 'GetLinkTemplates))
;;; #<VLA-OBJECT ILinkTemplates 04df12d0>
(setq DBLinkT (vla-Item DBLinksT "Link2"))
;;; #<VLA-OBJECT ILinkTemplate 04dff0d0>
(setq DBLinks1 (vlax-invoke-method DBConn 'GetLinks "Link2" nil nil
    actdoc))
;;; #<VLA-OBJECT ILinks 0f843ad0>
(setq DBLinks2 (vlax-invoke-method DBConn 'GetLinks DBLinkT
  (vlax-make-variant (vlax-safearray-fill (vlax-make-safearray
    vlax-vbLong '(0 . 0)) (list (vla-get-ObjectID
    (vlax-ename->vla-object (car (entsel))))))) 2 actdoc))
;;; #<VLA-OBJECT ILinks 0f8405f0>
(vlax-invoke-method DBLinks1 'Intersect DBLinks2)
;;; #<VLA-OBJECT ILinks 0c9332b0>
```

7.2.230. Метод *IntersectWith*

Объекты: **все графические объекты**, кроме **Pviewport** и **PolygonMesh**.

Описание: получение точек пересечения графических объектов или их продлений.

Аргументы:

- <второй объект> — графический VLA-объект, с которым производится поиск пересечения;
- <режим продления>:

  acExtendNone (0) — не продлевать;

  acExtendThisEntity (1) — продлевать только первый объект;

acExtendOtherEntity (2) — продлевать только второй объект;

acExtendBoth (3) — продлевать оба объекта.

Возврат: массив координат точек пересечения в виде варианта.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadCircle 0bf4aaf4>
(setq temp (vla-IntersectWith obj (vlax-ename->vla-object (car (entsel)))
    acExtendThisEntity))
;;; #<variant 8197 ...>
(vlax-safearray-get-u-bound (vlax-variant-value temp) 1)
;;; 5
;;; убедились, что массив не пуст; 5 -> 6 элементов (считая с 0)
;;; пустой массив вернет -1 (минус единицу)
(vlax-safearray->list (vlax-variant-value temp))
;;; (2673.5 187.356 0.0 2602.29 279.693 0.0)
;;; две точки пересечения -> (X1 Y1 Z1 X2 Y2 Z2)
```

7.2.231. Метод *IsConnected*

Объект: **DbConnect**.

Описание: определение наличия подключения к источнику данных.

Аргументы:

- <источник> — строка, имя источника.

Возврат:

- :vlax-true — есть;
- :vlax-false — нет.

```
 (setq DBConn (vla-GetInterfaceObject (vlax-get-acad-object)
              "CAO.DbConnect.16"))
;;; #<VLA-OBJECT IDbConnect 0f8184d0>
(vlax-invoke-method DBConn 'IsConnected "jet_dbsamples")
;;; :vlax-true
```

7.2.232. Метод *IsMergedCell*

Объект: **Table**.

Описание: проверка статуса объединенности ячеек.

Аргументы:

- <строка>, <колонка> — неотрицательные целые числа, базовая ячейка;
- <строка мин.>, <строка макс.>, <колонка мин.>, <колонка макс.> — неотрицательные целые числа, верхние и нижние границы диапазона поиска.

Возврат:

- `:vlax-true` — да;
- `:vlax-false` — нет.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0bf42854>
(vla-IsMergedCell obj 2 5 0 1 0 0)
;;; :vlax-true
```

7.2.233. Метод *IsRemoteFile*

Объект: **`Utility`**.

Описание: проверка того, является ли локальный файл копией полученной с удаленного узла. См. метод `GetRemoteFile`.

Аргументы:

- `<файл>` — строка, имя локального файла;
- `<ссылка>` — строка, URL.

Возврат:

- `:vlax-true` — да;
- `:vlax-false` — нет.

```
(setq obj (vla-get-Utility actdoc))
;;; #<VLA-OBJECT IAcadUtility 0bf43714>
(vla-IsRemoteFile obj "C:\\Doc ... PL_relead(0.9)[1].9).zip"
    "http://www.uniip.ru/lib/download/files/lisp/PL_relead(0.9).zip")
;;; :vlax-true
```

7.2.234. Метод *IsURL*

Объект: **`Utility`**.

Описание: проверка URL-ссылки на соответствие стандартам интернет-протоколов.

Аргументы:

- `<ссылка>` — строка, URL.

Возврат:

- `:vlax-true` — да;
- `:vlax-false` — нет.

```
(setq obj (vla-get-Utility actdoc))
;;; #<VLA-OBJECT IAcadUtility 0bf43714>
(vla-IsURL obj "http://www.uniip.ru/ ... /PL_relead(0.9).zip")
;;; :vlax-true
```

7.2.235. Метод *Item*

Объекты: **все семейства**, **Group**, **SelectionSet**.

Описание: получение элемента по индексу или имени.

Аргументы:

- <элемент> — строка или неотрицательное целое число.

Возврат: VLA-объект.

```
(setq obj (vla-get-Blocks actdoc))
;;; #<VLA-OBJECT IAcadBlocks 0bf41ac4>
(vla-Item obj "tb")
;;; #<VLA-OBJECT IAcadBlock2 0bf43304>
;;; по имени блока
(vla-Item obj 8)
;;; #<VLA-OBJECT IAcadBlock2 0bf432a4>
;;; по индексу в коллекции
```

7.2.236. Метод *LaunchBrowserDialog*

Объект: **Utility**.

Описание: запуск встроенного Web-браузера с параметрами.

Аргументы:

- <конечная страница> — символ, которому присваивается строка;
- <заголовок> — строка, заголовок окна браузера;
- <заголовок кнопки "OK"> — строка;
- <стартовая страница> — строка;
- <параметры> — строка (ключ в реестре, содержащий постоянные параметры);
- <разрешение выхода с подтверждением>:

  :vlax-true — разрешен;

  :vlax-false — запрещен.

Возврат:

- :vlax-true — выход с подтверждением;
- :vlax-false — выход с отказом.

```
(setq obj (vla-get-Utility actdoc))
;;; #<VLA-OBJECT IAcadUtility 0bf43714>
(vla-LaunchBrowserDialog obj 'temp "Пробный запуск" "Выход"
    "C:/Program Files/Autodesk/Autodesk DWF Viewer/startup.htm" ""
    :vlax-true)
;;; :vlax-true
temp
;;; "file:///C:/Program Files/Autodesk/Autodesk DWF Viewer/startup.htm"
```

7.2.237. Метод *ListArx*

Объект: **Application**.

Описание: получение списка загруженных ARX-приложений.

Аргументы: нет.

Возврат: список строк в виде варианта.

```
(setq obj (vlax-get-acad-object))
;;; #<VLA-OBJECT IAcadApplication 00b9d604>
(setq temp (vla-ListArx obj))
;;; #<variant 8200 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; ("acapp.arx" "acblock.arx" ... "wscommcntraccon.arx")
```

7.2.238. Метод *Load*

Объекты: **Linetypes**, **MenuGroups**.

Описание: загрузка группы адаптации интерфейса (из CUI-файла), группы меню (из MNU-файла) или типа линий (из LIN-файла).

Аргументы (различные случаи):

- **Linetypes**
  - <тип линий> — строка, имя типа линий;
  - <файл> — строка, имя файла.
- **MenuGroups**
  - <файл> — строка, имя файла;
  - [<базовое меню>]:

    :vlax-true — загрузка адаптации, как базовой, или меню, как базового;

    :vlax-false — загрузка адаптации, как частичной, или меню, как фрагментного (по умолчанию).

Возврат: созданный VLA-объект или nil.

```
(setq obj (vla-get-Linetypes actdoc))
;;; #<VLA-OBJECT IAcadLineTypes 0bb29624>
(vla-Load obj "DOT2" "acadiso.lin")
;;; загружен тип линий
(setq obj (vla-get-MenuGroups (vlax-get-acad-object)))
;;; #<VLA-OBJECT IAcadMenuGroups 01083718>
(vla-Load obj "dbcon.mns" :vlax-false)
;;; #<VLA-OBJECT IAcadMenuGroup 010d3574>
```

7.2.239. Метод *LoadArx*

Объект: **Application**.

Описание: загрузка ARX-приложения.

Аргументы:

♦ <файл> — строка, имя файла.

Возврат: nil.

```
(setq obj (vlax-get-acad-object))
;;; #<VLA-OBJECT IAcadApplication 00b9d604>
(vla-LoadArx obj "acapp.arx")
```

7.2.240. Метод *LoadDVB*

Объект: **Application**.

Описание: загрузка DVB-файла (проекта VBA).

Аргументы:

♦ <файл> — строка, имя файла.

Возврат: nil.

```
(setq obj (vlax-get-acad-object))
;;; #<VLA-OBJECT IAcadApplication 00b9d604>
(vla-LoadDVB obj "C:/Program ... Sample/VBA/drawline.dvb")
```

7.2.241. Метод *LoadShapeFile*

Объект: **Document**.

Описание: загрузка файла форм.

Аргументы:

♦ <файл> — строка, полное имя файла.

Возврат: nil.

```
(vla-LoadShapeFile actdoc "C:/Program Files/ ... /Fonts/MyShape.SHX")
```

7.2.242. Метод *MergeCells*

2005

Объект: **Table**.

Описание: объединение ячеек.

Аргументы:

- <строка мин.>, <строка макс.>, <колонка мин.>, <колонка макс.> — неотрицательные целые числа, верхние и нижние границы диапазона.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 1153c4c4>
(vla-MergeCells obj 0 1 1 1)
```

7.2.243. Метод *Mirror*

Объекты: **все графические объекты**.

Описание: зеркальное копирование относительно вертикальной плоскости, определяемой по двум точкам.

Аргументы:

- <точка 1>, <точка 2> — координаты трехмерных точек на оси, задающей плоскость симметрии, в виде варианта с типом безопасного массива.

Возврат: созданный VLA-объект.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcad3DFace 114c3f44>
(vla-Mirror obj (vlax-3D-point (getpoint)) (vlax-3D-point (getpoint)))
;;; #<VLA-OBJECT IAcad3DFace 11522584>
```

7.2.244. Метод *Mirror3D*

Объекты: **все графические объекты**.

Описание: зеркальное копирование относительно плоскости, определяемой по трем точкам.

Аргументы:

- <точка 1>, <точка 2>, <точка 3> — координаты трехмерных точек на оси симметрии в виде варианта с типом безопасного массива.

Возврат: созданный VLA-объект.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadEllipse 115401b4>
```

```
(vla-Mirror3D obj (vlax-3D-point (getpoint)) (vlax-3D-point (getpoint))
    (vlax-3D-point (getpoint)))
;;; #<VLA-OBJECT IAcadEllipse 0f067834>
```

7.2.245. Метод *Move*

Объекты: **все графические объекты**, **AttributeReference**.

Описание: перемещение вдоль заданного вектора.

Аргументы:

- <точка 1>, <точка 2> — координаты трехмерных точек начала и конца вектора перемещения в виде варианта с типом безопасного массива.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadLWPolyline 0b87ac24>
(vla-Move obj (vlax-3D-point (getpoint)) (vlax-3D-point (getpoint)))
```

7.2.246. Метод *MoveAbove*

Объект: **SortentsTable**.

Описание: помещение графических объектов поверх указанного путем изменения порядка следования.

Аргументы:

- <объекты> — вариант (безопасный массив) перемещаемых объектов;
- <цель> — объект, поверх которого нужно поместить.

Возврат: nil.

```
(setq dict (vla-GetExtensionDictionary (vla-get-ModelSpace actdoc)))
;;; #<VLA-OBJECT IAcadDictionary 0b87a894>
(setq obj (vla-GetObject dict "ACAD_SORTENTS"))
;;; #<VLA-OBJECT IAcadSortentsTable 0b87a7f4>
```

Примечание

Если объект SortentsTable не существует, то он должен быть создан при помощи метода AddObject.

```
(vla-MoveAbove obj (vlax-safearray-fill
    (vlax-make-safearray vlax-vbObject '(0 . 1))
    (list (vlax-ename->vla-object (car (entsel)))
          (vlax-ename->vla-object (car (entsel)))))
  (vlax-ename->vla-object (car (entsel))))
```

7.2.247. Метод *MoveBelow*

2005

Объект: **SortentsTable**.

Описание: помещение графических объектов под указанный.

Аргументы:

- <объекты> — вариант (безопасный массив) перемещаемых объектов;
- <цель> — объект, под который нужно поместить.

Возврат: nil.

```
(setq dict (vla-GetExtensionDictionary (vla-get-ModelSpace actdoc)))
;;; #<VLA-OBJECT IAcadDictionary 0b87a894>
(setq obj (vla-GetObject dict "ACAD_SORTENTS"))
;;; #<VLA-OBJECT IAcadSortentsTable 0b87a7f4>
```

Примечание

Если объект SortentsTable не существует, то он должен быть создан при помощи метода AddObject.

```
(vla-MoveBelow obj (vlax-safearray-fill
    (vlax-make-safearray vlax-vbObject '(0 . 0))
    (list (vlax-ename->vla-object (car (entsel)))))
  (vlax-ename->vla-object (car (entsel))))
```

7.2.248. Метод *MoveToBottom*

2005

Объект: **SortentsTable**.

Описание: помещение графических объектов вниз.

Аргументы:

- <объекты> — вариант (безопасный массив) перемещаемых объектов.

Возврат: nil.

```
(setq dict (vla-GetExtensionDictionary (vla-get-ModelSpace actdoc)))
;;; #<VLA-OBJECT IAcadDictionary 0b87a894>
(setq obj (vla-GetObject dict "ACAD_SORTENTS"))
;;; #<VLA-OBJECT IAcadSortentsTable 0b87a7f4>
```

Примечание

Если объект SortentsTable не существует, то он должен быть создан при помощи метода AddObject.

```
(vla-MoveToBottom obj (vlax-safearray-fill
    (vlax-make-safearray vlax-vbObject '(0 . 0))
    (list (vlax-ename->vla-object (car (entsel))))))
```

7.2.249. Метод *MoveToTop*

2005

Объект: **SortentsTable**.

Описание: помещение графических объектов наверх.

Аргументы:

- <объекты> — вариант (безопасный массив) перемещаемых объектов.

Возврат: nil.

```
(setq dict (vla-GetExtensionDictionary (vla-get-ModelSpace actdoc)))
;;; #<VLA-OBJECT IAcadDictionary 0b87a894>
(setq obj (vla-GetObject dict "ACAD_SORTENTS"))
;;; #<VLA-OBJECT IAcadSortentsTable 0b87a7f4>
```

Примечание

Если объект SortentsTable не существует, то он должен быть создан при помощи метода AddObject.

```
(vla-MoveToTop obj (vlax-safearray-fill
    (vlax-make-safearray vlax-vbObject '(0 . 0))
    (list (vlax-ename->vla-object (car (entsel))))))
```

7.2.250. Метод *New*

Объект: **Document**.

Описание: должен создавать новый документ в однодокументном режиме, но не может этого сделать функцией Visual LISP, поскольку для этого необходимо уничтожить текущий документ, являющийся средой исполнения функции.

Аргументы:

- <шаблон> — строка, имя шаблона.

Возврат: должен возвращать VLA-объект созданного документа.

```
(vla-New actdoc "acad.dwt")
;;; error: Automation Error.
;;; AutoCAD is unable to service automation request
```

7.2.251. Метод *NumCustomInfo*

Объект: **SummaryInfo**.

Описание: количество пользовательских записей в свойствах рисунка.

Аргументы: нет.

Возврат: неотрицательное целое число.

```
(setq obj (vla-get-SummaryInfo actdoc))
;;; #<VLA-OBJECT IAcadSummaryInfo 138f5374>
(vla-NumCustomInfo obj)
;;; 1
```

7.2.252. Метод *ObjectIDToObject*

Объект: **Document**.

Описание: получение объекта по его идентификатору (ID).

Аргументы:

- <идентификатор> — положительное целое число.

Возврат: VLA-объект.

```
(vla-ObjectIDToObject actdoc 2130046264)
;;; #<VLA-OBJECT IAcadLWPolyline 138f6e54>
```

7.2.253. Метод *Offset*

Объекты: **Arc**, **Circle**, **Ellipse**, **Line**, **LightweightPolyline**, **Polyline**, **Spline**, **XLine**.

Описание: создание эквидистантной копии объекта.

Аргументы:

- <смещение> — вещественное число.

Примечание

Направление создания копии: для отрезков — положительное значение влево от направления отрисовки примитива, для окружностей, эллипсов, круговых и эллиптических дуг — положительное значение наружу, для остальных — положительное значение вправо от направления отрисовки.

Возврат: созданный VLA-объект (в сложных случаях может быть несколько объектов) в варианте с типом безопасного массива.

Примечание

При выполнении данного метода может производиться автоматическое приведение типов копий объектов — например, `Ellipse` в `Spline`.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadSpline 138f8ee4>
(setq temp (vla-Offset obj -50))
;;; #<variant 8201 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (#<VLA-OBJECT IAcadSpline 13918e84>)
```

7.2.254. Метод *Open*

Объекты: **Document**, **Documents**.

Описание: открытие существующего документа, для Document только в однодокументном режиме (фактически для Document не работает, см. также метод New).

Аргументы:

- <файл> — строка, имя файла;
- [<режим>]:

  :vlax-true — только чтение;

  :vlax-false — редактирование (по умолчанию).
- [<пароль>] — строка, для доступа к защищенным файлам.

Возврат: VLA-объект.

```
(vla-Open (vla-get-Documents (vlax-get-acad-object))
     "D:/Tutor/Draw/Other.dwg" :vlax-true "")
;;; #<VLA-OBJECT IAcadDocument 0667c440>
```

7.2.255. Метод *PlotToDevice*

Объект: **Plot**.

Описание: печать на указанном устройстве вывода.

Аргументы:

- <устройство> — строка, имя устройства вывода.

Возврат:

- :vlax-true — отправка прошла успешно;
- :vlax-false — ошибка при отправке.

```
(setq obj (vla-get-plot actdoc))
;;; #<VLA-OBJECT IAcadPlot 138b058c>
(vla-PlotToDevice obj "TestPlot.pc3")
;;; :vlax-true
```

7.2.256. Метод *PlotToFile*

Объект: **Plot**.

Описание: печать в файл.

Аргументы:

- <файл> — строка, имя файла.
- <устройство> — строка, имя устройства вывода.

Возврат:

- `:vlax-true` — отправка прошла успешно;
- `:vlax-false` — ошибка при отправке.

```
(setq obj (vla-get-Plot actdoc))
;;; #<VLA-OBJECT IAcadPlot 138b058c>
(vla-PlotToFile obj "D:/temp/test.plt" "TestPlot.pc3")
;;; :vlax-true
```

7.2.257. Метод *PolarPoint*

Объект: **Utility**.

Описание: получение точки, отстоящей от другой точки на заданное расстояние в заданном направлении.

Аргументы:

- `<точка>` — координаты трехмерной точки в виде варианта с типом безопасного массива;
- `<угол>` — вещественное представление угла в радианах;
- `<дистанция>` — вещественное число.

Возврат: координаты трехмерной точки в виде варианта с типом безопасного массива.

```
(setq obj (vla-get-Utility actdoc))
;;; #<VLA-OBJECT IAcadUtility 0b276a04>
(setq temp (vla-PolarPoint obj (vlax-3D-point (getpoint)) (/ pi 8) -200))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (24.0183 111.231 0.0)
```

7.2.258. Метод *Prompt*

Объект: **Utility**.

Описание: вывод сообщения в командную строку.

Аргументы:

- `<сообщение>` — строка.

Возврат: `nil`.

```
(setq obj (vla-get-Utility actdoc))
;;; #<VLA-OBJECT IAcadUtility 0b276a04>
(vla-Prompt obj "\nЛюбая строка")
```

7.2.259. Метод *PurgeAll*

Объект: **Document**.

Описание: удаление из рисунка неиспользуемых слоев, описаний блоков и т. п.

Аргументы: нет. Возврат: nil.

```
(vla-PurgeAll actdoc)
```

7.2.260. Метод *PurgeFitData*

Объект: **Spline**.

Описание: удаление определяющих точек сплайна.

Аргументы: нет. Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadSpline 0b275814>
(vla-PurgeFitData obj)
```

7.2.261. Метод *PutRemoteFile*

Объект: **Utility**.

Описание: копирование локального файла, на внешний ресурс.

Аргументы:

- <ссылка> — строка, URL удаленного хранилища;
- <файл> — строка, имя локального файла.

Возврат: nil.

```
(setq obj (vla-get-Utility actdoc))
;;; #<VLA-OBJECT IAcadUtility 0c153854>
(vla-PutRemoteFile obj "http://server/folder/" "D:/Temp/test.dwg")
```

7.2.262. Метод *Quit*

Объект: **Application**.

Описание: выход из приложения.

Примечание

При выполнении данного метода все рисунки закрываются с предварительным запросом на сохранение (при необходимости).

Аргументы: нет. Возврат: nil.

```
(setq obj (vlax-get-acad-object))
;;; #<VLA-OBJECT IAcadApplication 00b9d604>
(vla-Quit obj)
```

7.2.263. Метод *RealToString*

Объект: **Utility**.

Описание: преобразование вещественного числа в строковое представление.

Аргументы:

- <число> — вещественное число;
- <единицы> — выходной формат представления (см. метод DistanceToReal);
- <точность> — целое число от 0 до 8, количество знаков после запятой.

Возврат: строка.

```
(setq obj (vla-get-Utility actdoc))
;;; #<VLA-OBJECT IAcadUtility 0b9ac8e4>
(vla-RealToString obj 123.4567 acDecimal 3)
;;; "123.457"
```

7.2.264. Метод *RecomputeTableBlock*

Объект: **Table**.

Описание: обновление таблицы.

Аргументы:

- <режим>:

  :vlax-true — принудительное обновление;

  :vlax-false — обновление по необходимости.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0b9c20b4>
(vla-RecomputeTableBlock obj :vlax-true)
```

7.2.265. Метод *Refresh*

Объекты: **LabelTemplate**, **Label**.

Описание: принудительное обновление.

Аргументы: нет. Возврат: nil.

```
(setq DBConn (vla-GetInterfaceObject (vlax-get-acad-object)
              "CAO.DbConnect.16"))
;;; #<VLA-OBJECT IDbConnect 0b665db0>
(setq DBLinksT (vlax-invoke DBConn 'GetLinkTemplates))
;;; #<VLA-OBJECT ILinkTemplates 0b665ed0>
(setq DBLinkT (vla-Item DBLinksT "Link2"))
;;; #<VLA-OBJECT ILinkTemplate 0b665680>
(setq DBLinks (vlax-invoke-method DBConn 'GetLinks DBLinkT
 (vlax-make-variant (vlax-safearray-fill (vlax-make-safearray
    vlax-vbLong '(0 . 0)) (list (vla-get-ObjectID
      (vlax-ename->vla-object (car (entsel))))))) 2 actdoc))
;;; #<VLA-OBJECT ILinks 0b665ae0>
(setq DBLink (vla-Item DBLinks 0))
;;; #<VLA-OBJECT ILink 0b661050>
(setq DBLabl (vlax-get-property DBLink 'Label))
;;; #<VLA-OBJECT ILabel 0b6654c0>
(vlax-invoke-method DBLabl 'Refresh)
```

7.2.266. Метод *RefreshPlotDeviceInfo*

Объекты: **Layout**, **PlotConfiguration**.

Описание: принудительное обновление настроек печати.

Аргументы: нет. Возврат: `nil`.

```
(setq obj (vla-get-ActiveLayout actdoc))
;;; #<VLA-OBJECT IAcadLayout 0b4af0d4>
(vla-RefreshPlotDeviceInfo obj)
```

7.2.267. Метод *Regen*

Объект: **Document**.

Описание: регенерация изображения.

Аргументы:

- `<режим>`:

  `acActiveViewport` (0) — только активное окно;

  `acAllViewports` (1) — все окна.

Возврат: `nil`.

```
(vla-Regen actdoc acAllViewports)
```

7.2.268. Метод *Reload*

Объекты: **Block**, **LabelTemplate**, **Label**.

Описание: принудительная перезагрузка внешних данных объекта.

Аргументы: нет. Возврат: nil.

```
(setq obj (vla-Item (vla-get-Blocks actdoc) "MyRef"))
;;; #<VLA-OBJECT IAcadBlock2 0b9f8324>
(vla-Reload obj)
```

7.2.269. Метод *ReloadLabels*

Объект: **DbConnect**.

Описание: принудительная перезагрузка ярлыков.

Аргументы:

- [<объекты>] — вариант, содержащий безопасный массив целых чисел — ID графических объектов;
- [<документ>] — VLA-объект документа, в контексте которого должен выполняться метод.

Примечание

При отсутствии аргумента метод выполняется в контексте текущего документа.

Возврат: nil.

Примечание

Для успешного выполнения данного метода внешняя база должна быть подключена.

```
(setq DBConn (vla-GetInterfaceObject (vlax-get-acad-object)
               "CAO.DbConnect.16"))
;;; #<VLA-OBJECT IDbConnect 0b665db0>
(setq DBLinksT (vlax-invoke DBConn 'GetLinkTemplates))
;;; #<VLA-OBJECT ILinkTemplates 0b665ed0>
(setq DBLinkT (vla-Item DBLinksT "Link2"))
;;; #<VLA-OBJECT ILinkTemplate 0b665680>
(setq GrObjs (vlax-make-variant (vlax-safearray-fill (vlax-make-safearray
    vlax-vbLong '(0 . 0)) (list (vla-get-ObjectID (vlax-ename->vla-object
    (car (entsel))))))))
;;; #<variant 8195 ...>
(vlax-invoke-method DBConn 'ReloadLabels GrObjs actdoc)
```

7.2.270. Метод *Remove*

Объект: **Dictionary**.

Описание: удаление объекта из словаря.

Аргументы:

♦ <ключ> — строка, идентификатор объекта.

Возврат: удаленный VLA-объект.

```
(setq obj (vla-Add (vla-get-Dictionaries actdoc) "MyTestDict"))
;;; #<VLA-OBJECT IAcadDictionary 0b9fa5f4>
(vla-Remove obj "TestKey")
;;; #<VLA-OBJECT IAcadTableStyle 0b9fa3c4>
```

7.2.271. Метод *RemoveCustomByIndex*

Объект: **SummaryInfo**.

Описание: удаление данных пользователя по номеру.

Аргументы:

♦ <номер> — неотрицательное целое число.

Возврат: nil.

```
(setq obj (vla-get-SummaryInfo actdoc))
;;; #<VLA-OBJECT IAcadSummaryInfo 0b9c2154>
(vla-RemoveCustomByIndex obj 0)
```

7.2.272. Метод *RemoveCustomByKey*

Объект: **SummaryInfo**.

Описание: удаление данных пользователя по ключу (имени записи).

Аргументы:

♦ <имя записи> — строка.

Возврат: nil.

```
(setq obj (vla-get-SummaryInfo actdoc))
;;; #<VLA-OBJECT IAcadSummaryInfo 0b9c2154>
(vla-RemoveCustomByKey obj "My Datum")
```

7.2.273. Метод *RemoveEntry*

Объект: **FileDependencies**.

Описание: удаление записи из таблицы зависимостей по индексу.

Аргументы:

- <индекс> — положительное целое число;
- <признак принудительного удаления>:

  :vlax-true — удалять в любом случае;

  :vlax-false — удалять только если счетчик пуст.

Возврат: nil.

```
(setq obj (vla-get-FileDependencies actdoc))
;;; #<VLA-OBJECT IAcadFileDependencies 0b469c64>
(vla-RemoveEntry obj 10 :vlax-true)
```

7.2.274. Метод *RemoveFromMenuBar*

Объект: **PopupMenu**.

Описание: удаление падающего меню из строки меню.

Аргументы: нет. Возврат: nil.

```
(setq obj (vla-Item (vla-get-Menus (vla-Item (vla-get-MenuGroups
    (vlax-get-acad-object)) "dbConnect")) "D&ata View"))
;;; #<VLA-OBJECT IAcadPopupMenu 01106688>
(vla-RemoveFromMenuBar obj)
```

7.2.275. Метод *RemoveItems*

Объекты: **Group**, **SelectionSet**.

Описание: удаление графических объектов из состава набора или группы.

Аргументы:

- <объекты> — массив объектов в виде варианта с типом безопасного массива.

Возврат: nil.

```
(setq obj (vla-get-ActiveSelectionSet actdoc))
;;; #<VLA-OBJECT IAcadSelectionSet 0b469054>
(vla-Clear obj)
(vla-SelectOnScreen obj nil nil)
(setq temp (mapcar '(lambda (a) (vlax-ename->vla-object (car a)))
    (list (entsel) (entsel) (entsel))))
;;; (#<VLA-OBJECT IAcadSpline 0b4da074> #<VLA-OBJECT IAcadLine ...)
(setq temp (vlax-safearray-fill
    (vlax-make-safearray vlax-vbObject '(0 . 2)) temp))
;;; #<safearray...>
(vla-RemoveItems obj temp)
```

7.2.276. Метод *RemoveMenuFromMenuBar*

Объект: **PopupMenus**.

Описание: удаление существующего меню из строки меню по номеру или имени.

Аргументы:

- <индекс> — неотрицательное целое число или строка.

Возврат: nil.

```
(setq obj (vla-get-Menus (vla-Item (vla-get-MenuGroups
    (vlax-get-acad-object)) "dbConnect")))
;;; #<VLA-OBJECT IAcadPopupMenus 0b9283d4>
(vla-RemoveMenuFromMenuBar obj "D&ata View")
```

7.2.277. Метод *Rename*

Объекты: **Dictionary**, **LayerStateManager**.

Описание: переименование элемента.

Аргументы:

- <старое имя>, <новое имя> — строки.

Возврат: nil.

```
(setq obj (vla-Item (vla-get-Dictionaries actdoc) "MyTestDict"))
;;; #<VLA-OBJECT IAcadDictionary 0c593b84>
(vla-Rename obj "MyTestRec" "NewNameTestRec")
```

7.2.278. Метод *RenameProfile*

Объект: **PreferencesProfiles**.

Описание: переименование профиля.

Аргументы:

- <старое имя>, <новое имя> — строки.

Возврат: nil.

```
(setq obj (vla-get-Profiles
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesProfiles 0b4df98c>
(vla-RenameProfile obj "My Profile" "New Name Profile")
```

7.2.279. Метод *Replace*

Объект: **Dictionary**.

Описание: переопределение элемента словаря.

Примечание

Если переопределяемый элемент в словаре не найден, то добавляется новый элемент.

Аргументы:

- <имя> — строка, имя записи словаря;
- <объект> — VLA-объект.

Возврат: `nil`.

```
(setq obj (vla-Item (vla-get-Dictionaries actdoc) "MyTestDict"))
;;; #<VLA-OBJECT IAcadDictionary 0c593b84>
(setq dict (vla-Item (vla-get-Dictionaries actdoc) "ACAD_MLINESTYLE"))
;;; #<VLA-OBJECT IAcadDictionary 0b89e844>
(vla-Replace obj "MyTestRec" (vla-Item dict 0))
```

7.2.280. Метод *ReselectSubRegion*

2005

Объект: **Table**.

Описание: обновление выборки.

Примечание

См. также метод `vla-SetSubSelection`.

Аргументы: нет. Возврат: `nil`.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0b4dc034>
(vla-ReselectSubRegion obj)
```

7.2.281. Метод *ResetBlock*

2006

Объекты: **BlockRef**, **MInsertBlock**.

Описание: возврат динамического блока к состоянию по умолчанию.

Аргументы: нет. Возврат: `nil`.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadBlockReference2 0b100d24>
(vla-ResetBlock obj)
```

7.2.282. Метод *ResetProfile*

Объект: **PreferencesProfiles**.

Описание: установка профилю значений по умолчанию.

Аргументы:

- <имя профиля> — строка.

Возврат: nil.

```
(setq obj (vla-get-Profiles
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesProfiles 0ba66fcc>
(vla-ResetProfile obj "New Name Profile")
```

7.2.283. Метод *Restore*

Объект: **LayerStateManager**.

Описание: восстановление свойств слоев из именованного состояния.

Аргументы:

- <состояние> — имя сохраненного состояния слоев.

Возврат: nil.

```
(setq obj (vla-GetInterfaceObject (vlax-get-acad-object)
    "AutoCAD.AcadLayerStateManager.16"))
;;; #<VLA-OBJECT IAcadLayerStateManager 0ba62364>
(vla-SetDatabase obj (vla-get-Database (vla-get-ModelSpace actdoc)))
;;; Инициализация объекта.
(vla-Restore obj "MyState")
```

7.2.284. Метод *Reverse*

Объект: **Spline**.

Описание: изменение направления отрисовки.

Аргументы: нет.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadSpline 0ba7cf44>
(vla-Reverse obj)
```

7.2.285. Метод *Rotate*

Объекты: **все графические объекты, AttributeReference**.

Описание: поворот вокруг вертикальной оси, задаваемой точкой в проекции.

Аргументы:

- <точка> — координаты трехмерной точки в виде варианта с типом безопасного массива;
- <угол поворота> — вещественное число, угол в радианах.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadLine 0ba80c14>
(vla-Rotate obj (vlax-3D-point (getpoint)) (/ pi 12))
```

7.2.286. Метод *Rotate3D*

Объекты: **все графические объекты**, **AttributeReference**.

Описание: поворот вокруг произвольной оси.

Аргументы:

- <точка 1>, <точка 2> — координаты трехмерных точек в виде варианта с типом безопасного массива;
- <угол поворота> — вещественное число, угол в радианах.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcad3DSolid 0ba809b4>
(vla-Rotate3D obj (vlax-3D-point (getpoint)) (vlax-3D-point (getpoint))
   (/ pi 12))
```

7.2.287. Метод *RunMacro*

Объект: **Application**.

Описание: выполнение макроса языка VBA.

Аргументы:

- <путь> — строка, путь к макросу.

Возврат: nil.

```
(setq obj (vlax-get-acad-object))
;;; #<VLA-OBJECT IAcadApplication 00b9d604>
(vla-LoadDVB obj "C:/Program ... Sample/VBA/drawline.dvb")
;;; загрузка проекта
(vla-RunMacro obj "Module1.Drawline")
```

7.2.288. Метод *Save*

Объекты: **Document**, **LayerStateManager**, **MenuGroup**.

Описание: сохранение объекта.

Аргументы (различные случаи):

- **Document** — нет.
- **LayerStateManager**
  - <имя> — строка, имя конфигурации;
  - <маска> — сохраняемые свойства слоев, см. свойство Mask.
- **MenuGroup**[1]
  - <тип файла>:

    acMenuFileCompiled (0) — компилированный файл;

    acMenuFileSource (1) — исходный текст.

Возврат: nil.

```
(vla-Save actdoc)
;;; сохранение документа
(setq obj (vla-GetInterfaceObject (vlax-get-acad-object)
    "AutoCAD.AcadLayerStateManager.16"))
;;; #<VLA-OBJECT IAcadLayerStateManager 0baa8fe4>
(vla-SetDatabase obj (vla-get-Database (vla-get-ModelSpace actdoc)))
(vla-Save obj "MyNewState" (+ acLsOn acLsFrozen acLsNewViewport))
;;; сохранение состояния слоев
(setq obj (vla-Item (vla-get-MenuGroups (vlax-get-acad-object))
    "dbConnect"))
;;; #<VLA-OBJECT IAcadMenuGroup 010fd3d4>
(vla-Save obj acMenuFileSource)
;;; сохранение меню
```

7.2.289. Метод *SaveAs*

Объекты: **Document**, **MenuGroup**.

Описание: сохранение объекта в файл с новым именем.

Аргументы (различные случаи):

- **Document**
  - <имя файла> — строка (расширение не требуется);
  - <тип файла> — см. свойство SaveAsType;
  - [<защита>] — VLA-объект шифровки рисунка.

---

[1] Не поддерживается в AutoCAD 2006.

- **MenuGroup**[2]
  - <имя файла> — строка (расширение не требуется);
  - <тип файла> — см. метод Save.

Возврат: nil.

```
(vla-SaveAs actdoc "D:/Temp/Test" ac2000_dxf)
;;; сохранение документа
(setq obj (vla-Item (vla-get-MenuGroups (vlax-get-acad-object))
    "dbConnect"))
;;; #<VLA-OBJECT IAcadMenuGroup 010fd3d4>
(vla-SaveAs obj "D:/Temp/Test" acMenuFileSource)
;;; сохранение меню
```

7.2.290. Метод *ScaleEntity*

Объекты: **все графические объекты**, **AttributeReference**.

Описание: пропорциональное масштабирование по всем осям.

Аргументы:

- <точка> — координаты трехмерной точки в виде варианта с типом безопасного массива;
- <масштабный коэффициент> — вещественное число.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadLine 0b99c6e4>
(vla-ScaleEntity obj (vlax-3D-point (getpoint)) 1.33)
```

7.2.291. Метод *SectionSolid*

Объект: **3DSolid**.

Описание: создание объекта типа Region путем сечения тела плоскостью, определяемой по трем точкам.

Аргументы:

- <точка 1>, <точка 2>, <точка 3> — координаты трехмерных точек в виде варианта с типом безопасного массива.

Возврат: созданный VLA-объект.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcad3DSolid 0b49b054>
```

[2] Не поддерживается в AutoCAD 2006.

```
(vla-SectionSolid obj (vlax-3D-point (getpoint))
    (vlax-3D-point (getpoint)) (vlax-3D-point (getpoint)))
;;; #<VLA-OBJECT IAcadRegion 0b99b1e4>
```

7.2.292. Метод *Select*

Объекты: **SelectionSet**, **Table**.

Описание: добавление графических объектов в набор или выбор ячейки таблицы.

Аргументы (различные случаи):

- **SelectionSet**
  - <режим>:

    acSelectionSetWindow (0) — рамка;

    acSelectionSetCrossing (1) — секущая рамка;

    acSelectionSetPrevious (3) — предыдущий;

    acSelectionSetLast (4) — последний;

    acSelectionSetAll (5) — все.
  - [<точка 1>], [<точка 2>] — координаты трехмерных точек в виде варианта с типом безопасного массива;
  - [<коды фильтров>] — безопасный массив целых чисел в виде варианта;
  - [<значения фильтров>] — безопасный массив типа "вариант" в виде варианта.
- **Table**
  - <исходная точка>, <вектор поиска> — координаты трехмерных точек в виде варианта с типом безопасного массива;
  - <ориентация>, <ширина прицела>, <высота прицела> — положительные вещественные числа (параметры прицела зарезервированы на будущее);
  - <режим>:
  - :vlax-true — разрешение на указание точки вне таблицы;
  - :vlax-false — запрет на указание точки вне таблицы.
  - <строка>, <колонка> — неотрицательные целые числа (вычисляются), координаты ячейки.

Возврат: nil.

```
(setq obj (vla-get-ActiveSelectionSet actdoc))
;;; #<VLA-OBJECT IAcadSelectionSet 0c548d94>
;;; получили безымянный набор
(vla-Clear obj)
;;; очистили его, на всякий случай (только для этого примера)
```

```
(vla-Select obj acSelectionSetAll nil nil (vlax-safearray-fill
    (vlax-make-safearray vlax-vbinteger '(0 . 1)) '(0 8))
  (vlax-safearray-fill (vlax-make-safearray vlax-vbvariant '(0 . 1))
    '("LINE" "LAYER3")))
;;; выбрали все отрезки, находящиеся на слое "LAYER3"
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0b9b3104>
(vla-Select obj (vlax-3D-point (getpoint))
 (vlax-3D-point '(0.0 0.0 1.0)) 0 20 5 :vlax-true 'row 'col)
row
;;; 2
col
;;; 3
```

7.2.293. Метод *SelectAtPoint*

Объект: **SelectionSet**.

Описание: добавление в набор графических объектов методом выбора точкой.

Аргументы:

- <точка> — координаты трехмерной точки в виде варианта с типом безопасного массива;
- [<коды фильтров>] — безопасный массив целых чисел в виде варианта;
- [<значения фильтров>] — вариант (безопасный массив) с данными типа "вариант".

Возврат: nil.

```
(setq obj (vla-get-ActiveSelectionSet actdoc))
;;; #<VLA-OBJECT IAcadSelectionSet 0b970494>
;;; получили безымянный набор
(vla-Clear obj)
;;; очистили его, на всякий случай (только для этого примера)
(vla-SelectAtPoint obj (vlax-3D-point (getpoint)) (vlax-safearray-fill
    (vlax-make-safearray vlax-vbinteger '(0 . 0)) '(0))
  (vlax-safearray-fill (vlax-make-safearray vlax-vbvariant '(0 . 0))
    '("LINE")))
```

7.2.294. Метод *SelectByPolygon*

Объект: **SelectionSet**.

Описание: добавление графических объектов в набор с помощью режима выбора.

Аргументы:

- <режим>:

  acSelectionSetFence (2) — линия;

  acSelectionSetWindowPolygon (6) — многоугольник;

  acSelectionSetCrossingPolygon (7) — секущий многоугольник.
- <массив точек> — координаты трехмерных точек в виде варианта с типом плоского безопасного массива;
- [<коды фильтров>] — безопасный массив целых чисел в виде варианта;
- [<значения фильтров>] — вариант (безопасный массив) с данными типа "вариант".

Возврат: nil.

```
(setq obj (vla-get-ActiveSelectionSet actdoc))
;;; #<VLA-OBJECT IAcadSelectionSet 0c548d94>
;;; получили безымянный набор
(vla-Clear obj)
;;; очистили его, на всякий случай (только для этого примера)
(vla-SelectByPolygon obj acSelectionSetWindowPolygon
 (vlax-safearray-fill (vlax-make-safearray vlax-vbDouble '(0 . 8))
   (append (getpoint) (getpoint) (getpoint)))
 (vlax-safearray-fill (vlax-make-safearray vlax-vbInteger '(0 . 0)) '(0))
 (vlax-safearray-fill (vlax-make-safearray vlax-vbVariant '(0 . 0))
   '("LINE")))
```

7.2.295. Метод *SelectOnScreen*

Объект: **SelectionSet**.

Описание: интерактивное добавление графических объектов в набор.

Аргументы:

- [<коды фильтров>] — безопасный массив целых чисел в виде варианта;
- [<значения фильтров>] — вариант (безопасный массив) с данными типа "вариант".

Возврат: nil.

```
(setq obj (vla-get-ActiveSelectionSet actdoc))
;;; #<VLA-OBJECT IAcadSelectionSet 0c548d94>
;;; получили безымянный набор
(vla-Clear obj)
;;; очистили его, на всякий случай (только для этого примера)
```

```
(vla-SelectOnScreen obj
 (vlax-safearray-fill (vlax-make-safearray vlax-vbInteger
  '(0 . 0)) '(0))
 (vlax-safearray-fill (vlax-make-safearray vlax-vbVariant
  '(0 . 0)) '("LINE")))
```

7.2.296. Метод *SelectSubRegion*

Объект: **Table**.

Описание: установка выборки в таблице указанием двух точек.

Аргументы:

- <точка 1>, <точка 2>, <направление>, <ориентация> — координаты трехмерных точек в виде варианта с типом безопасного массива, углы прямоугольника выбора и ориентация выборки;
- <режим выбора>:

  acTableSelectWindow (1) — рамка;

  acTableSelectCrossing (2) — секущая рамка.
- <режим возврата>:

  :vlax-true — разрешение на указание точки вне таблицы;

  :vlax-false — запрет на указание точки вне таблицы.
- <строка мин.>, <строка макс.>, <колонка мин.>, <колонка макс.> — символы, которым присваиваются индексы.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0b9a9f24>
(vla-SelectSubRegion obj (vlax-3D-point (getpoint))
    (vlax-3D-point (getpoint)) (vlax-3D-point '(0 1 0))
    (vlax-3D-point '(0 0 1)) acTableSelectWindow :vlax-true 'rowmin
     'rowmax 'colmin 'colmax)
rowmin
;;; 2
rowmax
;;; 4
colmin
;;; 0
colmax
;;; 2
```

7.2.297. Метод *SendCommand*

Объект: **Document**.

Описание: посылка выражения в командную строку AutoCAD.

Аргументы:

- <выражение> — строка.

Возврат: nil.

```
(vla-SendCommand actdoc "_.line 0,0 300,300  ") ;_ на конце 2 пробела
```

7.2.298. Метод *SendModelessOperationEnded* 2006

Объект: **Utility**.

Описание: индикация конца немодальных операций.

Аргументы:

- <контекст> — строка.

Возврат: nil.

```
(setq obj (vla-get-Utility actdoc))
;;; #<VLA-OBJECT IAcadUtility2 0b109f24>
(vla-SendModelessOperationEnded obj "some string")
```

7.2.299. Метод *SendModelessOperationStart* 2006

Объект: **Utility**.

Описание: индикация начала немодальных операций.

Аргументы:

- <контекст> — строка.

Возврат: nil.

```
(setq obj (vla-get-Utility actdoc))
;;; #<VLA-OBJECT IAcadUtility2 0b109f24>
(vla-SendModelessOperationStart obj "some string")
```

7.2.300. Метод *SetAlignment* 2005

Объекты: **Table**, **TableStyle**.

Описание: установка выравнивания содержимого ячеек таблицы.

Аргументы:

- <тип строк> — см. метод GetAlignment;
- <выравнивание>:

  acTopLeft (1) — верх, влево;

  acTopCenter (2) — верх, центр;

acTopRight (3) — верх, вправо;

acMiddleLeft (4) — середина, влево;

acMiddleCenter (5) — середина, центр;

acMiddleRight (6) — середина, вправо;

acBottomLeft (7) — низ, влево;

acBottomCenter (8) — низ, центр;

acBottomRight (9) — низ, вправо.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0bd9cc34>
(vla-SetAlignment obj acDataRow acTopRight)
```

7.2.301. Метод *SetAutoScale*

Объект: **Table**.

Описание: установка значения режима автомасштабирования содержимого ячейки.

Аргументы:

- <строка>, <колонка> — неотрицательные целые числа;
- <режим>:

  :vlax-true — включен;

  :vlax-false — выключен.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0bd9cc34>
(vla-SetAutoScale obj 2 1 :vlax-true)
```

7.2.302. Метод *SetBackgroundColor*

Объекты: **Table**, **TableStyle**.

Описание: установка значения цвета фона строк.

Аргументы:

- <тип строк> — указание типа строк таблицы, см. метод GetAlignment.
- <цвет> — объект типа AcCmColor.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0bd9cc34>
```

```
(setq color (vla-GetInterfaceObject (vlax-get-acad-object)
    "AutoCAD.AcCmColor.16"))
;;; #<VLA-OBJECT IAcadAcCmColor 0b66bce0>
;;; получение объекта AcCmColor, без настроек
(vla-SetBackgroundColor obj acDataRow color)
```

7.2.303. Метод *SetBackgroundColorNone*

Объекты: **Table**, **TableStyle**.

Описание: управление режимом отсутствия фона нефонового цвета строк.

Аргументы:

- <тип строк> — указание типа строк, см. метод GetAlignment.
- <режим>:

  :vlax-true — включен;

  :vlax-false — выключен.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0bd9cc34>
(vla-SetBackgroundColorNone obj acDataRow :vlax-true)
```

7.2.304. Метод *SetBitmaps*

Объект: **ToolbarItem**.

Описание: установка имен иконок для режимов обычных и крупных кнопок.

Аргументы:

- <малая>, <большая> — строки, имена файлов или ресурсов иконок.

Возврат: nil.

```
(setq obj (vla-Item (vla-Item (vla-get-Toolbars (vla-Item
 (vla-get-MenuGroups (vlax-get-acad-object)) "ACAD"))
 "TestToolbar") 0))
;;; #<VLA-OBJECT IAcadToolbarItem 010f4fbc>
(vla-SetBitmaps obj "testpict.bmp" "testpict_large.bmp")
```

7.2.305. Метод *SetBlockAttributeValue*

Объект: **Table**.

Описание: установка значения атрибута блока в ячейке.

Аргументы:

- <строка>, <колонка>, <ID атрибута> — неотрицательные целые числа;
- <значение> — строка.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0bdb8834>
(vla-SetBlockAttributeValue obj 1 3 (vla-get-ObjectID
    (vla-Item (vla-Item (vla-get-Blocks actdoc) "ee") 2)) "Some String")
```

7.2.306. Метод *SetBlockRotation*

2005

Объект: **Table**.

Описание: установка значения угла поворота блока в ячейке в радианах.

Аргументы:

- <строка>, <колонка> — неотрицательные целые числа;
- <угол поворота> — вещественное число, угол в радианах.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0bdb8834>
(vla-SetBlockRotation obj 1 3 (/ pi 3))
```

7.2.307. Метод *SetBlockScale*

2005

Объект: **Table**.

Описание: установка значения масштаба блока в ячейке.

Аргументы:

- <строка>, <колонка> — неотрицательные целые числа;
- <масштаб> — положительное вещественное число.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0bdb8834>
(vla-SetBlockScale obj 1 3 0.025)
```

7.2.308. Метод *SetBlockTableRecordId*

2005

Объект: **Table**.

Описание: установка идентификатора записи ячейки.

Аргументы:

- <строка>, <колонка> — неотрицательные целые числа;
- <идентификатор блока> — положительное целое число;
- <режим вписывания>:

  :vlax-true — включен;

  :vlax-false — выключен.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0b8089d4>
(vla-SetBlockTableRecordId obj 1 3 2130042424 :vlax-false)
```

7.2.309. Метод *SetBulge*

Объекты: **LightweightPolyline**, **Polyline**.

Описание: установка кривизны сегмента, актуально только для полилиний типа acSimplePoly.

Аргументы:

- <индекс> — неотрицательное целое, номер опрашиваемой вершины;
- <кривизна> — вещественное число (¼ тангенса угла дуги), положительные значения соответствуют направлению против часовой стрелки.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadLWPolyline 13d1a144>
(vla-SetBulge obj 3 0.25)
```

7.2.310. Метод *SetCellAlignment*

Объект: **Table**.

Описание: установка выравнивания содержимого ячейки.

Аргументы:

- <строка>, <колонка> — неотрицательные целые числа;
- <выравнивание> — см. метод GetAlignment.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 13d1d894>
(vla-SetCellAlignment obj 2 2 acMiddleCenter)
```

7.2.311. Метод *SetCellBackgroundColor* 2005

Объект: **Table**.

Описание: установка цвета фона ячейки.

Аргументы:

- <строка>, <колонка> — неотрицательные целые числа;
- <цвет> — объект типа AcCmColor.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 137bee94>
(setq color (vla-GetInterfaceObject (vlax-get-acad-object)
    "AutoCAD.AcCmColor.16"))
;;; #<VLA-OBJECT IAcadAcCmColor 137bfe40>
;;; получение объекта — AcCmColor, без настроек
(vla-SetCellBackgroundColor obj 2 2 color)
```

7.2.312. Метод *SetCellBackgroundColorNone* 2005

Объект: **Table**.

Описание: управление режимом отсутствия фона ячейки.

Аргументы:

- <строка>, <колонка> — неотрицательные целые числа.
- <режим>:

  :vlax-true — включен;

  :vlax-false — выключен.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 137bee94>
(vla-SetCellBackgroundColorNone obj 2 2 :vlax-false)
```

7.2.313. Метод *SetCellContentColor* 2005

Объект: **Table**.

Описание: установка цвета содержимого ячейки.

Аргументы:

- <строка>, <колонка> — неотрицательные целые числа;
- <цвет> — объект типа AcCmColor.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 137bee94>
(setq color (vla-GetInterfaceObject (vlax-get-acad-object)
    "AutoCAD.AcCmColor.16"))
;;; #<VLA-OBJECT IAcadAcCmColor 13d1a230>
;;; получение объекта AcCmColor, без настроек
(vla-SetCellContentColor obj 2 2 color)
```

7.2.314. Метод *SetCellGridColor*

2005

Объект: **Table**.

Описание: установка цвета рамки ячейки.

Аргументы:

- <строка>, <колонка> — неотрицательные целые числа;
- <грань> — см. метод GetCellGridColor.
- <цвет> — объект типа AcCmColor;

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0b938224>
(setq color (vla-GetInterfaceObject (vlax-get-acad-object)
    "AutoCAD.AcCmColor.16"))
;;; #<VLA-OBJECT IAcadAcCmColor 13cfe260>
;;; получение объекта AcCmColor, без настроек
(vla-SetCellGridColor obj 2 2 (+ acTopMask acBottomMask) color)
```

7.2.315. Метод *SetCellGridLineWeight*

2005

Объект: **Table**.

Описание: установка веса линии рамки ячейки.

Аргументы:

- <строка>, <колонка> — неотрицательные целые числа;
- <грань> — см. метод GetCellGridColor;
- <вес> — см. свойство DimensionLineWeight.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0bd29844>
(vla-SetCellGridLineWeight obj 2 2 acBottomMask acLnWt020)
```

7.2.316. Метод *SetCellGridVisibility*

2005

Объект: **Table**.

Описание: установка режима видимости рамки ячейки.

Аргументы:

- <строка>, <колонка> — неотрицательные целые числа;
- <грань> — см. метод GetCellGridColor;
- <режим>:

  :vlax-true — видима;

  :vlax-false — невидима.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0bd29844>
(vla-SetCellGridVisibility obj 2 2 acBottomMask :vlax-false)
```

7.2.317. Метод *SetCellTextHeight*

2005

Объект: **Table**.

Описание: установка высоты текста ячейки.

Аргументы:

- <строка>, <колонка> — неотрицательные целые числа;
- <высота> — положительное вещественное число.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0bd29844>
(vla-SetCellTextHeight obj 2 2 8)
```

7.2.318. Метод *SetCellTextStyle*

2005

Объект: **Table**.

Описание: установка стиля текста ячейки.

Аргументы:

- <строка>, <колонка> — неотрицательные целые числа;
- <стиль> — строка, имя стиля.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0bd29844>
(vla-SetCellTextStyle obj 2 2 "MyTextStyle")
```

7.2.319. Метод *SetCellType*

2005

Объект: **Table**.

Описание: установка типа ячейки.

Аргументы:

- <строка>, <колонка> — неотрицательные целые числа;
- <тип> — см. метод GetCellType.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0b938224>
(vla-SetCellType obj 2 2 acTextCell)
```

7.2.320. Метод *SetColor*

2005

Объект: **TableStyle**.

Описание: установка цвета ячеек таблицы.

Аргументы:

- <тип строк> — указание типа строк, см. метод GetAlignment;
- <цвет> — объект типа AcCmColor.

Возврат: nil.

```
(setq obj (vla-Item (vla-Item (vla-get-Dictionaries actdoc)
    "ACAD_TABLESTYLE") "test"))
;;; #<VLA-OBJECT IAcadTableStyle 0bd4a234>
(setq color (vla-GetInterfaceObject (vlax-get-acad-object)
    "AutoCAD.AcCmColor.16"))
;;; #<VLA-OBJECT IAcadAcCmColor 0bd4a1a0>
;;; получение объекта AcCmColor, без настроек
(vla-SetColor obj acDataRow color)
```

7.2.321. Метод *SetColorBookColor*

Объект: **AcCmColor**.

Описание: установка значения цвета из имеющегося альбома цветов.

Аргументы[3]:

- <имя альбома>, <имя цвета> — строки.

Возврат: nil.

```
(setq obj (vla-GetInterfaceObject (vlax-get-acad-object)
    "AutoCAD.AcCmColor.16"))
;;; #<VLA-OBJECT IAcadAcCmColor 0c420f10>
(vla-SetColorBookColor obj "ral classic" "RAL 1032")
```

7.2.322. Метод *SetColumnWidth*

2005

Объект: **Table**.

Описание: установка ширины колонки.

Аргументы:

- <колонка> — неотрицательное целое число;
- <ширина> — положительное вещественное число.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0c3f0d94>
(vla-SetColumnWidth obj 2 100)
```

7.2.323. Метод *SetContentColor*

2005

Объект: **Table**.

Описание: установка цвета ячеек.

Аргументы:

- <тип строк> — указание типа строк, см. метод GetAlignment;
- <цвет> — объект типа AcCmColor.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0c3f0d94>
(setq color (vla-GetInterfaceObject (vlax-get-acad-object)
    "AutoCAD.AcCmColor.16"))
;;; #<VLA-OBJECT IAcadAcCmColor 0c447c00>
;;; получение объекта AcCmColor, без настроек
(vla-SetContentColor obj acDataRow color)
```

[3] В официальной документации ошибка в порядке аргументов.

7.2.324. Метод *SetControlPoint*

Объект: **Spline**.

Описание: установка координат управляющей точки по номеру.

Аргументы:

- <индекс> — неотрицательное целое число, номер точки;
- <точка> — координаты трехмерной точки в виде варианта с типом безопасного массива.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadSpline 0bcb3f24>
(vla-SetControlPoint obj 2 (vlax-3D-point (getpoint)))
```

7.2.325. Метод *SetCustomByIndex*

Объект: **SummaryInfo**.

Описание: замена имеющейся пользовательской информации по номеру.

Аргументы:

- <индекс> — неотрицательное целое число;
- <ключ> и <значение> — строки.

Возврат: nil.

```
(setq obj (vla-get-SummaryInfo actdoc))
;;; #<VLA-OBJECT IAcadSummaryInfo 0bcb3d64>
(vla-SetCustomByIndex obj 0 "My New Datum" "New string")
```

7.2.326. Метод *SetCustomByKey*

Объект: **SummaryInfo**.

Описание: замена имеющейся пользовательской информации по ключу (имени поля).

Аргументы:

- <ключ> и <значение> — строки.

Возврат: nil.

```
(setq obj (vla-get-SummaryInfo actdoc))
#<VLA-OBJECT IAcadSummaryInfo 0bcb3d64>
(vla-SetCustomByKey obj "My New Datum" "Any string")
```

7.2.327. Метод *SetCustomScale*

Объекты: **Layout**, **PlotConfiguration**.

Описание: установка пользовательского масштаба как частное от деления чисел.

Аргументы:

- <числитель>, <знаменатель> — положительные вещественные числа.

Возврат: nil.

```
(setq obj (vla-get-ActiveLayout actdoc))
;;; #<VLA-OBJECT IAcadLayout 0be7b914>
(vla-SetCustomScale obj 2 500)
```

7.2.328. Метод *SetDatabase*

Объект: **LayerStateManager**.

Описание: связывание базы системы AutoCAD с объектом.

Аргументы:

- <база> — VLA-объект.

Возврат: nil.

```
(setq obj (vla-GetInterfaceObject (vlax-get-acad-object)
    "AutoCAD.AcadLayerStateManager.16"))
;;; #<VLA-OBJECT IAcadLayerStateManager 0b77b7c4>
(vla-SetDatabase obj (vla-get-Database (vla-get-ModelSpace actdoc)))
```

7.2.329. Метод *SetFieldId*

Объект: **Table**.

Описание: связывание идентификатора текстового объекта, содержащего поле, с ячейкой.

Аргументы:

- <строка>, <колонка>, <идентификатор> — неотрицательные целые числа.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0b6fa024>
(setq temp (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadLine 0be803f4>
(vla-SetFieldId obj 1 4 (vla-get-ObjectID temp))
```

7.2.330. Метод *SetFitPoint*

Объект: **Spline**.

Описание: установка координат определяющей точки по номеру.

Аргументы:

- <индекс> — неотрицательные целое число, номер точки;
- <точка> — координаты трехмерной точки в виде варианта с типом безопасного массива.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadSpline 0be82e04>
(vla-SetFitPoint obj 2 (vlax-3D-point (getpoint)))
```

7.2.331. Метод *SetFont*

Объект: **TextStyle**.

Описание: установка параметров текстового стиля.

Аргументы:

- <шрифт> — строка;
- <полужирный>, <наклонный>, <набор>, <характер и семейство> — см. метод GetFont.

Возврат: nil.

```
(setq obj (vla-get-ActiveTextStyle actdoc))
;;; #<VLA-OBJECT IAcadTextStyle 0be7a8a4>
(vla-SetFont obj "Book Antiqua" :vlax-false :vlax-false 0 18)
```

7.2.332. Метод *SetGridColor*

Объекты: **Table**, **TableStyle**.

Описание: установка цвета сетки ячеек.

Аргументы:

- <тип сетки> — см. метод GetFont;
- <тип строк> — указание типа строк, см. метод GetAlignment;
- <цвет> — объект типа AcCmColor.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0f74d564>
```

```
(setq color (vla-GetInterfaceObject (vlax-get-acad-object)
    "AutoCAD.AcCmColor.16"))
;;; #<VLA-OBJECT IAcadAcCmColor 0c447c00>
;;; получение объекта AcCmColor, без настроек
(vla-SetGridColor obj acHorzBottom acDataRow color)
```

7.2.333. Метод *SetGridLineWeight*

2005

Объекты: **Table**, **TableStyle**.

Описание: установка веса линий сетки ячеек.

Аргументы:

- <тип сетки> — указание типа границ, см. метод GetGridColor;
- <тип строк> — указание типа строк, см. метод GetAlignment;
- <вес> — см. метод GetGridLineWeight.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0f74d564>
(vla-SetGridLineWeight obj acHorzBottom acDataRow -2)
```

7.2.334. Метод *SetGridSpacing*

Объекты: **Viewport**, **PViewport**.

Описание: установка размеров сетки.

Аргументы:

- <ось X>, <ось Y> — положительные вещественные числа.

Возврат: nil.

```
(setq obj (vla-get-ActivePViewport actdoc))
;;; #<VLA-OBJECT IAcadPViewport2 0b7794d4>
(vla-SetGridSpacing obj 25 20)
```

7.2.335. Метод *SetGridVisibility*

2005

Объекты: **Table**, **TableStyle**.

Описание: установка режима отображения линий сетки ячеек.

Аргументы:

- <тип сетки> — указание типа границ, см. метод GetGridColor;
- <тип строк> — указание типа строк, см. метод GetAlignment;

- <режим>:

  :vlax-true — отображаются;

  :vlax-false — не отображаются.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0f74d564>
(vla-SetGridVisibility obj acHorzBottom acDataRow :vlax-false)
```

7.2.336. Метод *SetInvisibleEdge*

Объект: **3DFace**.

Описание: установка режима видимости кромки.

Аргументы:

- <индекс> — целое число от 0 до 3 (номер кромки грани);
- <режим>:

  :vlax-true — видимая;

  :vlax-false — невидимая.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcad3DFace 0beccfa4>
(vla-SetInvisibleEdge obj 2 :vlax-false)
```

7.2.337. Метод *SetLayoutsToPlot*

Объект: **Plot**.

Описание: задание вкладок для печати.

Аргументы:

- <список вкладок> — безопасный массив строк.

Возврат: nil.

```
(setq obj (vla-get-Plot actdoc))
;;; #<VLA-OBJECT IAcadPlot 0beee36c>
(setq temp (vlax-safearray-fill (vlax-make-safearray vlax-vbString
    '(0 . 1)) '("Layout2" "Layout4")))
(vla-SetLayoutsToPlot obj temp)
```

7.2.338. Метод *SetNames*

Объект: **AcCmColor**.

Описание: установка имени цвета в альбоме цветов (не проверяется).

Аргументы:

- <имя цвета>, <имя альбома> — строки.

Возврат: nil.

```
(setq obj (vla-GetInterfaceObject (vlax-get-acad-object)
    "AutoCAD.AcCmColor.16"))
;;; #<VLA-OBJECT IAcadAcCmColor 0c420f10>
(vla-SetNames obj "MyColor" "MyAlbum.acb")
```

7.2.339. Метод *SetPattern*

Объект: **Hatch**.

Описание: задание типа и образца штриховки.

Аргументы:

- <тип> — см. метод AddHatch;
- <образец> — строка, имя образца.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadHatch 0c4d2964>
(vla-SetPattern obj acHatchPatternTypePreDefined "ANSI34")
```

7.2.340. Метод *SetProjectFilePath*

Объект: **PreferencesFiles**.

Описание: установка пути к файлам проекта.

Аргументы:

- <имя проекта>, <путь> — строки.

Возврат: nil.

```
(setq obj (vla-get-Files
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesFiles2 0c4d5910>
(vla-SetProjectFilePath obj "Book Tutorial" "D:\\Tutor")
```

7.2.341. Метод *SetRelativeDrawOrder*

2005

Объект: **SortentsTable**.

Описание: установка относительного порядка сортировки примитивов.

Примечание

Для реализации действия метода может потребоваться дополнительная регенерация экрана.

Аргументы:

- <порядок> — безопасный массив, содержащий объекты в обратном порядке.

Возврат: nil.

```
(setq dict (vla-GetExtensionDictionary (vla-get-ModelSpace actdoc)))
;;; #<VLA-OBJECT IAcadDictionary 0c4dd204>
(setq obj (vla-GetObject dict "ACAD_SORTENTS"))
;;; #<VLA-OBJECT IAcadSortentsTable 0c4de6a4>
(vla-SetRelativeDrawOrder obj (vlax-safearray-fill (vlax-make-safearray
  vlax-vbObject '(0 . 1)) (list (vlax-ename->vla-object (car (entsel)))
    (vlax-ename->vla-object (car (entsel))))))
```

7.2.342. Метод *SetRGB*

Объект: **AcCmColor**.

Описание: установка значений цветовых компонент в цвете.

Аргументы:

- <красный>, <зеленый>, <синий> — целые числа от 0 до 255.

Возврат: nil.

```
(setq obj (vla-GetInterfaceObject (vlax-get-acad-object)
    "AutoCAD.AcCmColor.16"))
;;; #<VLA-OBJECT IAcadAcCmColor 0c4dc2e0>
(vla-SetRGB obj 0 115 255)
```

7.2.343. Метод *SetRowHeight*

2005

Объект: **Table**.

Описание: установка высоты строки.

Аргументы:

- <номер строки> — неотрицательное целое число;
- <высота> — положительное вещественное число.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0c4dcd64>
(vla-SetRowHeight obj 2 12)
```

7.2.344. Метод *SetSnapSpacing*

Объекты: **PViewport**, **Viewport**.

Описание: установка параметров режима SNAP (ШАГ).

Аргументы:

- <ось X>, <ось Y> — положительные вещественные числа.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadPViewport2 0c525074>
(vla-SetSnapSpacing obj 100 100)
```

7.2.345. Метод *SetSubSelection*

Объект: **Table**.

Описание: установка индексов выборки.

Аргументы:

- <строка мин.>, <строка макс.>, <колонка мин.>, <колонка макс.> — неотрицательные целые числа.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0c520bc4>
(vla-SetSubSelection obj 0 2 1 2)
```

7.2.346. Метод *SetText*

Объект: **Table**.

Описание: установка текста в ячейке.

Аргументы:

- <строка>, <колонка> — положительные целые числа;
- <текст> — строка.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0c520bc4>
(vla-SetText obj 1 2 "Строка")
```

7.2.347. Метод *SetTextHeight*

2005

Объекты: **Table**, **TableStyle**.

Описание: установка высоты содержимого ячеек таблицы.

Аргументы:

- <тип строк> — см. метод GetTextHeight;
- <высота> — положительное вещественное число.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0c520bc4>
(vla-SetTextHeight obj acDataRow 7.5)
```

7.2.348. Метод *SetTextRotation*

2005

Объект: **Table**.

Описание: установка ориентации текста в ячейке.

Аргументы:

- <строка>, <колонка> — неотрицательные целые числа;
- <ориентация> — см. метод GetTextRotation.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0c520bc4>
(vla-SetTextRotation obj 1 2 1)
```

7.2.349. Метод *SetTextStyle*

2005

Объекты: **Table**, **TableStyle**.

Описание: установка текстового стиля ячеек таблицы.

Аргументы:

- <тип строк> — см. метод GetTextStyle;
- <стиль> — строка, имя стиля.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0c520bc4>
(vla-SetTextStyle obj acDataRow "Standard")
```

7.2.350. Метод *SetVariable*

Объект: **Document**.

Описание: установка значения системной переменной.

Аргументы:

- <имя> — строка, имя переменной;
- <значение> — корректный тип определяется в зависимости от переменной.

Возврат: nil.

```
(vla-SetVariable actdoc "cmdecho" 0)
```

7.2.351. Метод *SetView*

Объект: **Viewport**.

Описание: установка вида.

Аргументы:

- <вид> — VLA-объект.

Возврат: nil.

```
(setq obj (vla-get-ActiveViewport actdoc))
;;; #<VLA-OBJECT IAcadViewport 0c52e424>
(setq temp (vla-Item (vla-get-Views actdoc) "v2"))
;;; #<VLA-OBJECT IAcadView2 0c52fc04>
(vla-SetView obj temp)
```

7.2.352. Метод *SetWeight*

Объект: **Spline**.

Описание: установка веса управляющей точки по номеру. Метод неприменим к рациональным сплайнам.

Аргументы:

- <индекс> — неотрицательное целое число, номер точки;
- <вес> — положительное вещественное число.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadSpline 0bf76b34>
(vla-SetWeight obj 3 5)
```

7.2.353. Метод *SetWidth*

Объекты: **LightweightPolyline**, **Polyline**.

Описание: установка начальной и конечной ширины сегмента.

Аргументы:

- <индекс> — неотрицательное целое, номер сегмента (с нуля);
- <начало>, <конец> — положительные вещественные числа.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadLWPolyline 0bf691a4>
(vla-SetWidth obj 3 8 30)
```

7.2.354. Метод *SetWindowToPlot*

Объекты: **Layout**, **PlotConfiguration**.

Описание: установка координат окна, которое будет выводиться на принтер.

Аргументы:

- <первый>, <второй> — координаты углов в виде безопасного массива.

Возврат: nil.

```
(setq obj (vla-get-ActiveLayout actdoc))
;;; #<VLA-OBJECT IAcadLayout 0bfa4ae4>
(setq p1 (getpoint "\nПервая точка: ") p2 (getpoint "\nВторая точка: "))
;;; (4712.25 -534.517 0.0)
(vla-SetWindowToPlot obj (vlax-safearray-fill (vlax-make-safearray
 vlax-vbDouble '(0 . 1)) (list (car p1) (cadr p1))) (vlax-safearray-fill
 (vlax-make-safearray vlax-vbDouble '(0 . 1)) (list (car p2) (cadr p2))))
```

7.2.355. Метод *SetXData*

Объекты: **все графические объекты**, **AttributeReference**, **Block**, **Dictionary**, **DimStyle**, **Group**, **Layer**, **Linetype**, **PlotConfigurations**, **RegisteredApplication**, **TextStyle**, **UCS**, **View**, **Viewport**, **XRecord**.

Описание: запись расширенных данных примитива, ассоциированных с зарегистрированным приложением.

Аргументы:

- <тип> — безопасный массив целых чисел, DXF-кодов;
- <значения> — безопасный массив типа "вариант".

Примечание

Структура данных должна соответствовать требованиям, предъявляемым к расширенным данным.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadLine 0b858eb4>
(regapp "TestApp") ;_ регистрируем приложение
(vla-SetXData obj
  (vlax-safearray-fill (vlax-make-safearray vlax-vbInteger '(0 . 2))
     '(1001 1000 1040))
  (vlax-safearray-fill (vlax-make-safearray vlax-vbVariant '(0 . 2))
     '("TestApp" "Description" 12.34)))
```

7.2.356. Метод *SetXRecordData*

Объект: **XRecord**.

Описание: помещение данных в запись словаря.

Аргументы:

- <тип> — безопасный массив целых чисел, DXF-кодов;
- <значения> — безопасный массив типа "вариант".

Примечание

Структура данных должна соответствовать требованиям, предъявляемым к данным, помещаемым в записи словаря.

Возврат: nil.

```
(setq obj (vla-AddXRecord (vla-Item (vla-get-Dictionaries actdoc)
    "MyTestDict") "MyTestXRec")) ;_ добавляем запись в словарь
;;; #<VLA-OBJECT IAcadXRecord 0b92df74>
(vla-SetXRecordData obj
  (vlax-safearray-fill (vlax-make-safearray vlax-vbInteger '(0 . 1))
     '(2 140))
  (vlax-safearray-fill (vlax-make-safearray vlax-vbVariant '(0 . 2))
     '("Description" 12.34)))
```

7.2.357. Метод *Show*

Объекты: **LabelTemplate**, **Label**.

Описание: установка режима видимости. Метод, обратный методу `Hide`.

Аргументы: нет. Возврат: `nil`.

```
(setq DBConn (vla-GetInterfaceObject (vlax-get-acad-object)
               "CAO.DbConnect.16"))
;;; #<VLA-OBJECT IDbConnect 0f8184d0>
(setq DBLinksT (vlax-invoke DBConn 'GetLinkTemplates))
;;; #<VLA-OBJECT ILinkTemplates 0f83c510>
(setq DBLinkT (vla-Item DBLinksT "Link2"))
;;; #<VLA-OBJECT ILinkTemplate 0f83c010>
(setq DBLinks (vlax-invoke-method DBConn 'GetLinks DBLinkT
 (vlax-make-variant (vlax-safearray-fill (vlax-make-safearray
    vlax-vbLong '(0 . 0)) (list (vla-get-ObjectID
      (vlax-ename->vla-object (car (entsel))))))) 2 actdoc))
;;; #<VLA-OBJECT ILinks 0f83cf40>
(setq DBLink (vla-Item DBLinks 0))
;;; #<VLA-OBJECT ILink 0f83ccf0>
(setq DBLabl (vlax-get-property DBLink 'Label))
;;; #<VLA-OBJECT ILabel 0f838820>
(vlax-invoke-method DBLabl 'Show)
```

7.2.358. Метод *SliceSolid*

Объект: **3DSolid**.

Описание: создание объекта типа `3DSolid` путем сечения тела плоскостью, определяемой по трем точкам.

Аргументы:

- `<точка 1>`, `<точка 2>`, `<точка 3>` — координаты трехмерных точек в виде варианта с типом безопасного массива;
- `<режим>` — признак сохранения части, негативной относительно плана:

  `:vlax-true` — сохранить;

  `:vlax-false` — удалить.

Возврат: созданный VLA-объект или `nil`, если режим сохранения `:vlax-false`.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcad3DSolid 0bf644f4>
(vla-SliceSolid obj (vlax-3D-point (getpoint))
    (vlax-3D-point (getpoint)) (vlax-3D-point (getpoint)) :vlax-true)
;;; #<VLA-OBJECT IAcad3DSolid 0bf6cc54>
```

7.2.359. Метод *Split*

Объект: **Viewport**.

Описание: деление видового экрана на части.

Аргументы:

- <способ>:

  acViewport2Horizontal (0) — горизонтально на 2;

  acViewport2Vertical (1) — вертикально на 2;

  acViewport3Left (2) — на 3 с большим экраном слева;

  acViewport3Right (3) — на 3 с большим экраном справа;

  acViewport3Horizontal (4) — горизонтально на 3;

  acViewport3Vertical (5) — вертикально на 3;

  acViewport3Above (6) — на 3 с большим экраном сверху;

  acViewport3Below (7) — на 3 с большим экраном снизу;

  acViewport4 (8) — на 4.

Возврат: nil.

```
(setq obj (vla-get-ActiveViewport actdoc))
;;; #<VLA-OBJECT IAcadViewport 0c4cf3e4>
(vla-Split obj acViewport3Right)
(vla-put-ActiveViewport actdoc obj)
;;; активация нового состояния
```

7.2.360. Метод *StartBatchMode*

Объект: **Plot**.

Описание: запуск печати в пакетном режиме.

Примечание

Метод оставлен для совместимости, может быть изъят в следующих версиях.

Аргументы:

- <индекс> — положительное целое число, номер задания в очереди.

Возврат: nil.

```
(setq obj (vla-get-Plot actdoc))
;;; #<VLA-OBJECT IAcadPlot 0bdbcdac>
(vla-StartBatchMode obj 2)
```

7.2.361. Метод *StartUndoMark*

Объект: **Document**.

Описание: установка начальной метки отката. См. метод EndUndoMark.

Аргументы: нет. Возврат: nil.

```
(vla-StartUndoMark actdoc)
```

7.2.362. Метод *Subtract*

Объект: **Links**.

Описание: вычитание связей одной коллекции из другой (как множеств).

Аргументы:

- <связи> — VLA-объект, коллекция связей.

Возврат: VLA-объект.

```
(setq DBConn (vla-GetInterfaceObject (vlax-get-acad-object)
              "CAO.DbConnect.16"))
;;; #<VLA-OBJECT IDbConnect 0b665db0>
(setq DBLinksT (vlax-invoke DBConn 'GetLinkTemplates))
;;; #<VLA-OBJECT ILinkTemplates 0b665ed0>
(setq DBLinkT (vla-Item DBLinksT "Link2"))
;;; #<VLA-OBJECT ILinkTemplate 0b665680>
(setq DBLinks1 (vlax-invoke-method DBConn 'GetLinks "Link2" nil nil
    actdoc))
;;; #<VLA-OBJECT ILinks 0b670770>
(setq DBLinks2 (vlax-invoke-method DBConn 'GetLinks DBLinkT
  (vlax-make-variant (vlax-safearray-fill (vlax-make-safearray
    vlax-vbLong '(0 . 0)) (list (vla-get-ObjectID
    (vlax-ename->vla-object (car (entsel))))))) 2 actdoc))
;;; #<VLA-OBJECT ILinks 0b672fd0>
(vlax-invoke-method DBLinks1 'Subtract DBLinks2)
;;; #<VLA-OBJECT ILinks 0b6702f0>
```

7.2.363. Метод *SwapOrder*

Объект: **SortentsTable**.

Описание: смена порядка двух графических объектов.

Аргументы:

- <объект 1>, <объект 2> — VLA-объекты.

Возврат: nil.

```
(setq dict (vla-GetExtensionDictionary (vla-get-ModelSpace actdoc)))
;;; #<VLA-OBJECT IAcadDictionary 057f4d14>
(setq obj (vla-GetObject dict "ACAD_SORTENTS"))
;;; #<VLA-OBJECT IAcadSortentsTable 057f31a4>
(vla-SwapOrder obj (vlax-ename->vla-object (car (entsel)))
          (vlax-ename->vla-object (car (entsel))))
(vla-Regen actdoc acAllViewports)
```

7.2.364. Метод *SyncModelView*

2005

Объект: **PViewport**.

Описание: синхронизация содержимого видового экрана с уже ассоциированным видом.

Аргументы: нет. Возврат: `nil`.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadPViewport2 0be2b2d4>
(vla-SyncModelView obj)
```

7.2.365. Метод *TransformBy*

Объекты: **все графические объекты**, **AttributeReference**.

Описание: преобразование объекта по матрице преобразования размером 4×4.

Примечание

Не все объекты могут принимать любые матрицы трансформации.

Аргументы:

- <матрица> — вариант с типом безопасного массива 4×4.

Возврат: `nil`.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadLine 145c0a74>
(setq temp (vlax-tmatrix '((0 -1 0 80) (1 0 0 90) (0 0 1 0) (0 0 0 1))))
;;; #<variant 8197 ...>
(vla-TransformBy obj temp)
```

7.2.366. Метод *TranslateCoordinates*

Объект: **Utility**.

Описание: преобразование координат точки из одной ПСК в другую.

Аргументы:

- `<точка>` — координаты трехмерной точки в виде варианта с типом безопасного массива;
- `<из ПСК>`, `<в ПСК>` — целочисленные коды тех ПСК, между которыми выполняется преобразование:

  `acWorld` (0) — мировая;

  `acUCS` (1) — пользовательская;

  `acDisplayDCS` (2) — экранная;

  `acPaperSpaceDCS` (3) — экранная листа;

  `acOCS` (4) — объектная.
- `<режим>` — признак интерпретации, как вектора перемещения:

  `:vlax-true` — вектор;

  `:vlax-false` — точка.
- `[<нормаль>]` — объектная нормаль, координаты трехмерной точки в виде варианта с типом безопасного массива.

Возврат: координаты трехмерной точки в виде варианта с типом безопасного массива.

```
(setq obj (vla-get-Utility actdoc))
;;; #<VLA-OBJECT IAcadUtility 145c05b4>
(setq temp (vla-TranslateCoordinates obj (vlax-3D-point (getpoint)) acUCS
    acPaperSpaceDCS :vlax-false))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (379.382 -93.5118 -943.384)
```

7.2.367. Метод *Union*

Объект: **Links**.

Описание: объединение связей из двух коллекций.

Аргументы:

- `<связи>` — VLA-объект, коллекция связей.

Возврат: VLA-объект.

```
(setq DBConn (vla-GetInterfaceObject (vlax-get-acad-object)
               "CAO.DbConnect.16"))
;;; #<VLA-OBJECT IDbConnect 0b665db0>
(setq DBLinksT (vlax-invoke DBConn 'GetLinkTemplates))
;;; #<VLA-OBJECT ILinkTemplates 0b665ed0>
```

```
(setq DBLinkT (vla-Item DBLinksT "Link2"))
;;; #<VLA-OBJECT ILinkTemplate 0b665680>
(setq DBLinks1 (vlax-invoke-method DBConn 'GetLinks "Link2" nil nil
    actdoc))
;;; #<VLA-OBJECT ILinks 0b670770>
(setq DBLinks2 (vlax-invoke-method DBConn 'GetLinks DBLinkT
  (vlax-make-variant (vlax-safearray-fill (vlax-make-safearray
    vlax-vbLong '(0 . 0)) (list (vla-get-ObjectID
    (vlax-ename->vla-object (car (entsel))))))) 2 actdoc))
;;; #<VLA-OBJECT ILinks 0b672fd0>
(vlax-invoke-method DBLinks1 'Union DBLinks2)
;;; #<VLA-OBJECT ILinks 0b672e10>
```

7.2.368. Метод *Unload*

Объекты: **Block**, **MenuGroup**.

Описание: выгрузка объекта.

Аргументы: нет. Возврат: `nil`.

```
(setq obj (vla-Item (vla-get-Blocks actdoc) "MyRef"))
;;; #<VLA-OBJECT IAcadBlock2 145c4864>
(vla-Unload obj)
;;; выгрузка внешней ссылки
(setq obj (vla-Item (vla-get-MenuGroups (vlax-get-acad-object)) "MyDop"))
;;; #<VLA-OBJECT IAcadMenuGroup 010fed64>
(vla-Unload obj)
;;; выгрузка группы меню (2005) или группы адаптации (2006)
```

7.2.369. Метод *UnloadArx*

Объект: **Application**.

Описание: выгрузка ARX-приложения.

Аргументы:

- `<приложение>` — строка, имя файла.

Возврат: `nil`.

```
(setq obj (vlax-get-acad-object))
;;; #<VLA-OBJECT IAcadApplication 00b9d604>
(vla-UnloadArx obj "whohas.arx")
```

7.2.370. Метод *UnloadDVB*

Объект: **Application**.

Описание: выгрузка DVB-файла (проект VBA).

Аргументы:

- <файл> — строка, имя файла (символ "/" не допускается).

Возврат: nil.

```
(setq obj (vlax-get-acad-object))
;;; #<VLA-OBJECT IAcadApplication 00b9d604>
(vla-UnloadDVB obj "C:\\Program ... Sample\\VBA\\drawline.dvb")
```

7.2.371. Метод *UnmergeCells*

2005

Объект: **Table**.

Описание: разъединение ячеек.

Аргументы:

- <строка мин.>, <строка макс.>, <колонка мин.>, <колонка макс.> — неотрицательные целые числа, верхние и нижние границы диапазона.

Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 145e0894>
(vla-UnmergeCells obj 0 1 1 1)
```

7.2.372. Метод *Update*

Объекты: **все графические объекты**, **Application**, **AttributeReference**.

Описание: актуализация объекта в соответствии с текущим состоянием.

Аргументы: нет. Возврат: nil.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadText2 0c4c0034>
(vla-Update obj)
```

7.2.373. Метод *UpdateEntry*

Объект: **FileDependencies**.

Описание: актуализация записи объекта в таблице зависимостей по индексу.

Аргументы:

- <индекс> — положительное целое число.

Возврат: nil.

```
(setq obj (vla-get-FileDependencies actdoc))
;;; #<VLA-OBJECT IAcadFileDependencies 0c4c0a14>
(vla-UpdateEntry obj 3)
```

7.2.374. Метод *Wblock*

Объект: **Document**.

Описание: экспорт объектов набора в новый документ.

Аргументы:

- <файл> — строка, полное имя создаваемого или заменяемого файла;
- <набор> — VLA-объект.

Возврат: nil.

```
(setq temp (vla-get-ActiveSelectionSet actdoc))
;;; #<VLA-OBJECT IAcadSelectionSet 0c4c0f64>
(vla-Clear temp)
(vla-SelectOnScreen temp)
(vla-Wblock actdoc "D:/Temp/export.dwg" temp)
```

7.2.375. Метод *ZoomAll*

Объект: **Application**.

Описание: соответствует команде ZOOM (ПОКАЗАТЬ) с опцией **All** (Все).

Аргументы: нет. Возврат: nil.

```
(setq obj (vlax-get-acad-object))
;;; #<VLA-OBJECT IAcadApplication 00b9d604>
(vla-ZoomAll obj)
```

7.2.376. Метод *ZoomCenter*

Объект: **Application**.

Описание: соответствует команде ZOOM (ПОКАЗАТЬ) с опцией **Center** (Центр).

Аргументы:

- <точка> — координаты трехмерной точки в виде варианта с типом безопасного массива;
- <размер> — положительное вещественное число, высота экрана.

Возврат: `nil`.

```
(setq obj (vlax-get-acad-object))
;;; #<VLA-OBJECT IAcadApplication 00b9d604>
(vla-ZoomCenter obj (vlax-3D-point (getpoint)) 2000)
```

7.2.377. Метод *ZoomExtents*

Объект: **Application**.

Описание: соответствует команде ZOOM (ПОКАЗАТЬ) с опцией **Extents** (Границы).

Аргументы: нет. Возврат: `nil`.

```
(setq obj (vlax-get-acad-object))
;;; #<VLA-OBJECT IAcadApplication 00b9d604>
(vla-ZoomExtents obj)
```

7.2.378. Метод *ZoomPickWindow*

Объект: **Application**.

Описание: соответствует интерактивной команде ZOOM (ПОКАЗАТЬ) с опцией **Window** (Рамка) и запросом двух точек.

Аргументы: нет. Возврат: `nil`.

```
(setq obj (vlax-get-acad-object))
;;; #<VLA-OBJECT IAcadApplication 00b9d604>
(vla-ZoomPickWindow obj)
```

7.2.379. Метод *ZoomPrevious*

Объект: **Application**.

Описание: соответствует команде ZOOM (ПОКАЗАТЬ) с опцией **Previous** (Предыдущий).

Аргументы: нет. Возврат: `nil`.

```
(setq obj (vlax-get-acad-object))
;;; #<VLA-OBJECT IAcadApplication 00b9d604>
(vla-ZoomPrevious obj)
```

7.2.380. Метод *ZoomScaled*

Объект: **Application**.

Описание: соответствует команде ZOOM (ПОКАЗАТЬ) с опцией **Scale** (Масштаб).

Аргументы:

- <масштаб> — положительное вещественное число;
- <режим> — тип масштаба:

  acZoomScaledAbsolute (0) — абсолютный;

  acZoomScaledRelative (1) — относительный;

  acZoomScaledRelativePSpace (2) — относительно листа.

Возврат: nil.

```
(setq obj (vlax-get-acad-object))
;;; #<VLA-OBJECT IAcadApplication 00b9d604>
(vla-ZoomScaled obj (/ 1.0 100) acZoomScaledRelativePSpace)
```

7.2.381. Метод *ZoomWindow*

Объект: **Application**.

Описание: соответствует команде ZOOM (ПОКАЗАТЬ) с опцией **Window** (Рамка).

Аргументы:

- <точка 1> и <точка 2> — координаты трехмерной точки в виде варианта с типом безопасного массива.

Возврат: nil.

```
(setq obj (vlax-get-acad-object))
;;; #<VLA-OBJECT IAcadApplication 00b9d604>
(vla-ZoomWindow obj (vlax-3D-point (getpoint))
   (vlax-3D-point (getpoint)))
```

ГЛАВА 8

Свойства ActiveX

Данная глава логически связана с предыдущей, в которой разбирались методы ActiveX. Здесь мы рассмотрим функции Visual LISP, реализующие доступ к свойствам элементов объектной модели.

8.1. Дополнительные соглашения

Стиль изложения *глав 7* и *8* идентичен, поэтому на данную главу распространяются дополнительные соглашения предыдущей главы.

8.2. Функции свойств

В этот класс входят функции, позволяющие извлекать и изменять значения свойств объектов AutoCAD. Под свойством объекта понимаются не только свойства в принятом для CAD-систем смысле, такие как цвет или тип линий, но и объекты или их семейства, для которых исходный объект является контейнером. Во всех таких функциях первый аргумент — это тот VLA-объект, со свойством которого производится манипуляция. Некоторые свойства имеют атрибут "read only" (только чтение) и, соответственно, не могут быть изменены.

Обращение к свойствам объектов через интерфейс ActiveX возможно тремя способами: либо через функции `vlax-get-property` и `vlax-put-property`, либо через семейство именных функций свойств, либо с использованием функций `vlax-get` и `vlax-put`. Названия всех именных функций LISP этого типа формируются по единому правилу: **`vla-<действие>-<свойство>`**, где: `vla-` — общий префикс, `<действие>` — `get` (получить) или `put` (установить), `<свойство>` — имя свойства. Например, имя функции для получения цвета объекта будет `VLA-get-Color`, а для установки слоя объекта — `VLA-put-Layer`. Как уже упоминалось, кроме свойств в общепринятом смысле, таких как цвет или тип линий, функции данного семейст-

ва могут возвращать объекты или семейства объектов, для которых исходный объект является контейнером. Например, функция VLA-get-Layers возвращает семейство (строго говоря, указатель на семейство) слоев того документа, к которому она была применена. Вызов всех именных функций, обращающихся к свойствам, производится унифицированно, в следующем формате:

```
(vla-get-<свойство> <VLA-объект>)[1]
(vla-put-<свойство> <VLA-объект> <значение>)
```

Следует учитывать, что обращение к свойствам объектов вышеупомянутыми способами не полностью эквивалентно. Функции vlax-get-property и vlax-put-property более универсальны и позволяют обращаться к свойствам объектов не только AutoCAD, но и любых других приложений, поддерживающих технологию ActiveX (т. е. доступ к своей объектной модели), **без импорта соответствующих библиотек типов**. Естественно, для этого необходимо знать имена свойств объектов, к которым производится обращение и правила их использования. Приведенный ниже пошаговый пример получает указатель на объект Excel.Application (предварительно запущенный), "добирается" до листов и делает активным второй из них (подразумевается, что листов в книге не меньше двух).

```
(setq ExcelApp (vlax-get-object "Excel.Application"))
;;; #<VLA-OBJECT _Application 1555b0f4>
(setq ActBook (vlax-get-property ExcelApp 'ActiveWorkbook))
;;; #<VLA-OBJECT _Workbook 1279e7a4>
(setq Sheets (vlax-get-property ActBook 'Worksheets))
;;; #<VLA-OBJECT Sheets 127d9d84>
(setq Sheet2 (vlax-get-property Sheets 'Item 2))
;;; #<VLA-OBJECT _Worksheet 15521954>
(vlax-invoke-method Sheet2 'Activate)
;;; :vlax-true
```

Подробнее о работе с внешними приложениями *см. разд. 10.4*.

Функции vlax-get и vlax-put формально не документированы. Поэтому при их использовании нельзя быть ни в чем уверенными. Возможно, в очередной версии AutoCAD они окажутся изъятыми. Тем не менее в последних версиях (2002—2006) они есть, и их можно применять наравне с другими способами. Использование этих функций позволяет несколько упростить исходные тексты и сделать их более понятными, за счет уменьшения количества преобразований типов данных. Проиллюстрируем это на простом примере — сравним обращение к свойству AngleVertex двумя способами:

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDim3PointAngular 0acf2cd4>
```

[1] Кроме свойства StatusId.

```
(setq temp (vla-get-AngleVertex obj))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (132.785 116.232 0.0)
;;; Получили координаты точки через именную функцию...
(vlax-get obj 'AngleVertex)
;;; (132.785 116.232 0.0)

;;; а так это выглядит при использовании vlax-get
;;; Теперь изменение того же свойства:
(vla-put-AngleVertex obj (vlax-3d-point (getpoint)))
;;; Именная функция...
(vlax-put obj 'AngleVertex (getpoint))
;;; и функция vlax-put
```

Хорошо видно, что вызов `vlax-get` аналогичен вызову `vlax-get-property`, а `vlax-put`, соответственно, — `vlax-put-property`. Основным отличием является то, что в качестве аргументов и возвращаемых значений `vlax-get` и `vlax-put` фигурируют типы данных AutoLISP, а не ActiveX. Фактически эти функции производят автоматическое преобразование типов данных в нужный формат. Несмотря на такое неоспоримое удобство, использование этих функций не может быть рекомендовано ввиду неясности их дальнейшей судьбы.

В описании функций, реализующих свойства, мы используем следующую форму изложения. Сначала даются названия объектов, которые обладают этим свойством, затем приводится назначение и возможные варианты возвращаемых значений. Часто фигурируют зарезервированные константы технологии ActiveX. Завершается описание каждого свойства примером его использования.

В рамках данной книги рассматриваются свойства объектов только базовой версии AutoCAD. Для вертикальных приложений, в особенности разработанных непосредственно фирмой Autodesk, доступ к свойствам (например, к свойству `RoofLine` объекта `AecDbWall`) осуществляется по правилам доступа к свойствам объектов другого приложения.

8.2.1. Свойство *Action*

Объект: **`SecurityParams`**.

Описание: признаки цифровой подписи и шифрования. Возможные значения (могут суммироваться):

- 0 — не установлено (только чтение);
- `ACADSECURITYPARAMS_ENCRYPT_DATA` (1) — шифрование рисунка;
- `ACADSECURITYPARAMS_ENCRYPT_PROPS` (2) — шифрование свойств рисунка;
- `ACADSECURITYPARAMS_SIGN_DATA` (16) — наличие цифровой подписи;
- `ACADSECURITYPARAMS_ADD_TIMESTAMP` (32) — наличие отметки времени в подписи.

```
(setq obj (vla-GetInterfaceObject
            (vlax-get-acad-object) "AutoCAD.SecurityParams.16"))
;;; #<VLA-OBJECT IAcadSecurityParams 0a97c854>
(vla-get-Action obj)
;;; 0
(vla-put-Action obj ACADSECURITYPARAMS_SIGN_DATA)
```

8.2.2. Свойство *Active*

Объект: **Document**.

Описание: признак активности документа. Только чтение. Возможные значения:

- `:vlax-true` — документ активен;
- `:vlax-false` — документ неактивен.

```
(setq obj (vla-Item (vla-get-Documents (vlax-get-acad-object)) 0))
;;; #<VLA-OBJECT IAcadDocument 01043f90>
(vla-get-Active obj)
;;; :vlax-false
```

8.2.3. Свойство *ActiveDimStyle*

Объект: **Document**.

Описание: текущий размерный стиль. Возможные значения: VLA-объект размерного стиля.

```
(vla-get-ActiveDimStyle actdoc)
;;; #<VLA-OBJECT IAcadDimStyle 0a9b3f64>
(vla-put-ActiveDimStyle actdoc
  (vla-Item (vla-get-DimStyles actdoc) "New"))
```

8.2.4. Свойство *ActiveDocument*

Объект: **Application**.

Описание: текущий документ. Возможные значения: VLA-объект документа.

Примечание

При переключении активного документа выполнение функции приостанавливается до возвращения к тому документу, из которого она была вызвана.

```
(vla-get-ActiveDocument (vlax-get-acad-object))
;;; #<VLA-OBJECT IAcadDocument 0ab7c980>
(vla-put-ActiveDocument
  (vlax-get-acad-object)
  (vla-Item (vla-get-Documents (vlax-get-acad-object)) 0))
```

8.2.5. Свойство *ActiveLayer*

Объект: **Document**.

Описание: текущий слой. Возможные значения: VLA-объект слоя.

```
(vla-get-ActiveLayer actdoc)
;;; #<VLA-OBJECT IAcadLayer2 02461954>
(vla-put-ActiveLayer actdoc
  (vla-Item (vla-get-Layers actdoc) "Other"))
```

8.2.6. Свойство *ActiveLayout*

Объект: **Document**.

Описание: активная вкладка. Возможные значения: VLA-объект вкладки листа или вкладки **Model** (Модель).

```
(vla-get-ActiveLayout actdoc)
;;; #<VLA-OBJECT IAcadLayout 02460414>
(vla-put-ActiveLayout actdoc
  (vla-Item (vla-get-Layouts actdoc) "Other"))
```

8.2.7. Свойство *ActiveLinetype*

Объект: **Document**.

Описание: текущий тип линий (может изменяться только на загруженный в текущий рисунок тип линий). Возможные значения: VLA-объект типа линий.

```
(vla-get-ActiveLinetype actdoc)
;;; #<VLA-OBJECT IAcadLineType 0e7398a4>
(vla-put-ActiveLinetype actdoc
  (vla-Item (vla-get-Linetypes actdoc) "phantom"))
```

8.2.8. Свойство *ActiveProfile*

Объект: **PreferencesProfiles**.

Описание: текущий профиль. Возможные значения: строка с именем профиля.

```
(setq obj (vla-get-Profiles
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesProfiles 02459d8c>
(vla-get-ActiveProfile obj)
;;; "<<Unnamed Profile>>"
(vla-put-ActiveProfile obj "My Profile")
```

8.2.9. Свойство *ActivePViewport*

Объект: **Document**.

Описание: текущий видовой экран в пространстве листа *(см. также свойство ActiveViewport)*. Возможные значения: VLA-объект видового экрана.

Примечание

Переключение видовых экранов возможно только на вкладке листа в режиме **MODEL** (МОДЕЛЬ).

```
(vla-get-ActivePViewport actdoc)
;;; #<VLA-OBJECT IAcadPViewport2 00fd7884>
(vla-put-ActivePViewport actdoc
  (vla-Item (vla-get-PaperSpace actdoc) 1))
```

8.2.10. Свойство *ActiveSelectionSet*

Объект: **Document**.

Описание: текущий набор объектов чертежа. Только чтение. Возможные значения: VLA-объект набора.

```
(vla-get-ActiveSelectionSet actdoc)
;;; #<VLA-OBJECT IAcadSelectionSet 02447c34>
```

8.2.11. Свойство *ActiveSpace*

Объект: **Document**.

Описание: текущее пространство. Соответствует системной переменной TILEMODE. Возможные значения:

- `acPaperSpace` (0) — пространство листа;
- `acModelSpace` (1) — пространство модели (активна вкладка **Model** (Модель)).

```
(vla-get-ActiveSpace actdoc)
;;; 1
(vla-put-ActiveSpace actdoc acPaperSpace)
```

8.2.12. Свойство *ActiveTextStyle*

Объект: **Document**.

Описание: текущий текстовый стиль. Возможные значения: VLA-объект стиля.

```
(vla-get-ActiveTextStyle actdoc)
;;; #<VLA-OBJECT IAcadTextStyle 0f053ba4>
```

```
(vla-put-ActiveTextStyle actdoc
  (vla-Item (vla-get-TextStyles actdoc) "Other"))
```

8.2.13. Свойство *ActiveUCS*

Объект: **Document**.

Описание: текущая система координат (может изменяться только на существующую в рисунке систему координат). Возможные значения: VLA-объект ПСК.

```
(vla-get-ActiveUCS actdoc)
;;; #<VLA-OBJECT IAcadUCS 0ada40f4>
(vla-put-ActiveUCS actdoc
  (vla-Item (vla-get-UserCoordinateSystems actdoc) "Other"))
```

8.2.14. Свойство *ActiveViewport*

Объект: **Document**.

Описание: активный видовой экран в пространстве модели *(см. также ActivePViewport)*. Возможные значения: VLA-объект видового экрана.

```
(vla-get-ActiveViewport actdoc)
;;; #<VLA-OBJECT IAcadViewport 0adc14d4>
(vla-put-ActiveViewport actdoc
  (vla-Item (vla-get-Viewports actdoc) 1))
```

8.2.15. Свойство *ADCInsertUnitsDefaultSource*

Объект: **PreferencesUser**.

Описание: единицы измерения по умолчанию для источника, при вставке. Соответствует системной переменной INSUNITSDEFSOURCE. Возможные значения:

- `acInsertUnitsUnitless` (0) — не установлены;
- `acInsertUnitsInches` (1) — дюймы;
- `acInsertUnitsFeet` (2) — футы;
- `acInsertUnitsMiles` (3) — мили;
- `acInsertUnitsMillimeters` (4) — миллиметры;
- `acInsertUnitsCentimeters` (5) — сантиметры;
- `acInsertUnitsMeters` (6) — метры;
- `acInsertUnitsKilometers` (7) — километры;
- `acInsertUnitsMicroinches` (8) — микродюймы;
- `acInsertUnitsMils` (9) — милы;

- acInsertUnitsYards (10) — ярды;
- acInsertUnitsAngstroms (11) — ангстремы;
- acInsertUnitsNanometers (12) — нанометры;
- acInsertUnitsMicrons (13) — микроны;
- acInsertUnitsDecimeters (14) — дециметры;
- acInsertUnitsDecameters (15) — декаметры;
- acInsertUnitsHectometers (16) — гектометры;
- acInsertUnitsGigameters (17) — гигаметры;
- acInsertUnitsAstronomicalUnits (18) — астрономические единицы;
- acInsertUnitsLightYears (19) — световые годы;
- acInsertUnitsParsecs (20) — парсеки.

```
(setq obj (vla-get-User
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesUser 0efe9f88>
(vla-get-ADCInsertUnitsDefaultSource obj)
;;; 4
(vla-put-ADCInsertUnitsDefaultSource obj acInsertUnitsGigameters)
```

8.2.16. Свойство *ADCInsertUnitsDefaultTarget*

Объект: **PreferencesUser**.

Описание: единицы измерения по умолчанию для целевого файла, при вставке. Соответствует системной переменной INSUNITSDEFTARGET. Возможные значения: см. ADCInsertUnitsDefaultSource.

```
(setq obj (vla-get-User
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesUser 0efe9f88>
(vla-get-ADCInsertUnitsDefaultTarget obj)
;;; 4
(vla-put-ADCInsertUnitsDefaultTarget obj acInsertUnitsGigameters)
```

8.2.17. Свойство *AffectsGraphics*

Объект: **FileDependency**.

Описание: влияние внешнего файла на отображение графики. Только чтение. Возможные значения:

- :vlax-true — влияет;
- :vlax-false — не влияет.

```
(setq obj (vla-Item (vla-get-FileDependencies actdoc) 1))
;;; #<VLA-OBJECT IAcadFileDependency 0ad86c74>
(vla-get-AffectsGraphics obj)
;;; :vlax-true
```

8.2.18. Свойство *Algorithm*

Объект: **SecurityParams**.

Описание: алгоритм шифрования рисунка. Возможные значения:

- 0 — не установлен;
- ACADSECURITYPARAMS_ALGID_RC4 (26625) — установлен.

```
(setq obj (vla-GetInterfaceObject (vlax-get-acad-object)
           "AutoCAD.SecurityParams.16"))
;;; #<VLA-OBJECT IAcadSecurityParams 0a97c854>
(vla-get-Algorithm obj)
;;; 0
(vla-put-Algorithm obj ACADSECURITYPARAMS_ALGID_RC4)
```

8.2.19. Свойство *Alignment*

Объекты: **Attribute**, **AttributeRef**, **Text**.

Описание: выравнивание для однострочного текста, атрибута и описания атрибута. Возможные значения:

- acAlignmentLeft (0) — левое;
- acAlignmentCenter (1) — по центру;
- acAlignmentRight (2) — правое;
- acAlignmentAligned (3) — выравненное;
- acAlignmentMiddle (4) — посередине;
- acAlignmentFit (5) — вписанное;
- acAlignmentTopLeft (6) — верхнее левое;
- acAlignmentTopCenter (7) — верхнее по центру;
- acAlignmentTopRight (8) — верхнее правое;
- acAlignmentMiddleLeft (9) — среднее левое;
- acAlignmentMiddleCenter (10) — среднее по центру;
- acAlignmentMiddleRight (11) — среднее правое;
- acAlignmentBottomLeft (12) — нижнее левое;

- `acAlignmentBottomCenter` (13) — нижнее по центру;
- `acAlignmentBottomRight` (14) — нижнее правое.

Примечание

Если производится такое изменение выравнивания, при котором требуется дополнительная точка, не определенная у данного примитива (как в примере), то в качестве такой точки используется точка (0 0). *См. также свойство* `TextAlignmentPoint`.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadText2 0af2ce44>
(vla-get-Alignment obj)
;;; 0
(vla-put-Alignment obj acAlignmentMiddleCenter)
```

8.2.20. Свойство *AlignmentPointAcquisition*

Объект: **`PreferencesDrafting`**.

Описание: режим отображения векторов слежения при включенном режиме отслеживания. Возможные значения:

- `acAlignPntAcquisitionAutomatic` (0) — автоматический;
- `acAlignPntAcquisitionShiftToAcquire` (1) — при нажатии клавиши <Shift>.

```
(setq obj (vla-get-Drafting
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesDrafting 0aa5cba8>
(vla-get-AlignmentPointAcquisition obj)
;;; 0
(vla-put-AlignmentPointAcquisition obj
    acAlignPntAcquisitionShiftToAcquire)
```

8.2.21. Свойство *AllowedValues*

Объект: **`DynamicBlockReferenceProperty`**.

Описание: допустимые значения свойства динамического блока. Только чтение. Возможные значения: список значений в виде варианта, содержащего безопасный массив.

```
(setq obj (vlax-safearray-get-element (vlax-variant-value
    (vla-GetDynamicBlockProperties (vlax-ename->vla-object
    (car (entsel))))) 0))
;;; #<VLA-OBJECT IAcadDynamicBlockReferenceProperty 092e0444>
(setq temp (vla-get-AllowedValues obj))
;;; #<variant 8204 ...>
```

```
(mapcar 'vlax-variant-value (vlax-safearray->list
    (vlax-variant-value temp)))
;;; (30.0 40.0 50.0 51.6762)
```

8.2.22. Свойство *AllowLongSymbolNames*

Объект: **DatabasePreferences**.

Описание: разрешение на использование полного набора символов для имен файлов и длины имен более 31 символа. Соответствует системной переменной EXTNAMES. Возможные значения:

- `:vlax-true` — разрешает;
- `:vlax-false` — запрещает.

```
(setq obj (vla-get-Preferences actdoc))
;;; #<VLA-OBJECT IAcadDatabasePreferences 0a96ca75>
(vla-get-AllowLongSymbolNames obj)
;;; :vlax-true
(vla-put-AllowLongSymbolNames obj :vlax-false)
```

8.2.23. Свойство *AltFontFile*

Объект: **PreferencesFiles**.

Описание: альтернативный шрифт. Возможные значения: строка с именем шрифта. При изменении не проверяет наличие устанавливаемого шрифта.

```
(setq obj (vla-get-Files
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesFiles2 00fe3c40>
(vla-get-AltFontFile obj)
;;; "simplex.shx"
(vla-put-AltFontFile obj "txt.shx")
```

8.2.24. Свойство *AltRoundDistance*

Объекты: **DimAligned**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**.

Описание: округление альтернативных единиц примитива. Переопределяет значение, установленное системной переменной DIMALTRND. Возможные значения: неотрицательное вещественное число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimAligned 0ae19584>
```

```
(vla-get-AltRoundDistance obj)
;;; 0.0
(vla-put-AltRoundDistance obj 2.0)
```

8.2.25. Свойство *AltSuppressLeadingZeros*

Объекты: **DimAligned**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**.

Описание: подавление ведущих нулей альтернативных единиц примитива. Переопределяет значение, установленное системной переменной DIMALTZ. Возможные значения:

- `:vlax-true` — подавлять;
- `:vlax-false` — не подавлять.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimAligned 0ae19584>
(vla-get-AltSuppressLeadingZeros obj)
;;; :vlax-false
(vla-put-AltSuppressLeadingZeros obj :vlax-true)
```

8.2.26. Свойство *AltSuppressTrailingZeros*

Объекты: **DimAligned**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**.

Описание: подавление хвостовых нулей альтернативных единиц примитива. Переопределяет значение, установленное системной переменной DIMALTZ. Возможные значения:

- `:vlax-true` — подавлять;
- `:vlax-false` — не подавлять.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimAligned 0ae19584>
(vla-get-AltSuppressTrailingZeros obj)
;;; :vlax-false
(vla-put-AltSuppressTrailingZeros obj :vlax-true)
```

8.2.27. Свойство *AltSuppressZeroFeet*

Объекты: **DimAligned**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**.

Описание: подавление "футовых" нулей альтернативных единиц примитива. Переопределяет значение, установленное системной переменной DIMALTZ.

Возможные значения:

- `:vlax-true` — подавлять;
- `:vlax-false` — не подавлять.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimAligned 0ae19584>
(vla-get-AltSuppressZeroFeet obj)
;;; :vlax-true
(vla-put-AltSuppressZeroFeet obj :vlax-false)
```

8.2.28. Свойство *AltSuppressZeroInches*

Объекты: **DimAligned**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**.

Описание: подавление "дюймовых" нулей альтернативных единиц примитива. Переопределяет значение, установленное системной переменной DIMALTZ. Возможные значения:

- `:vlax-true` — подавлять;
- `:vlax-false` — не подавлять.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimAligned 0ae19584>
(vla-get-AltSuppressZeroInches obj)
;;; :vlax-true
(vla-put-AltSuppressZeroInches obj :vlax-false)
```

8.2.29. Свойство *AltTabletMenuFile*

Объект: **PreferencesFiles**.

Описание: альтернативный файл планшетного меню. Возможные значения: строка с именем файла или пустая строка. Не проверяет наличие устанавливаемого файла.

```
(setq obj (vla-get-Files
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesFiles2 00fe3c40>
(vla-get-AltTabletMenuFile obj)
;;; ""
(vla-put-AltTabletMenuFile obj "mymenu.mnu")
```

8.2.30. Свойство *AltTextPrefix*

Объекты: **DimAligned**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**.

Описание: текстовый префикс альтернативных единиц примитива. Переопределяет значение, установленное системной переменной DIMAPOST. Возможные значения: строка.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimAligned 0ae19584>
(vla-get-AltTextPrefix obj)
;;; ""
(vla-put-AltTextPrefix obj "В метрах: ")
```

8.2.31. Свойство *AltTextSuffix*

Объекты: **DimAligned**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**.

Описание: текстовый суффикс альтернативных единиц примитива. Переопределяет значение, установленное системной переменной DIMAPOST. Возможные значения: строка.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimAligned 0ae19584>
(vla-get-AltTextSuffix obj)
;;; ""
(vla-put-AltTextSuffix obj " м")
```

8.2.32. Свойство *AltTolerancePrecision*

Объекты: **DimAligned**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**.

Описание: точность допуска альтернативных единиц примитива. Переопределяет значение, установленное системной переменной DIMTDEC. Возможные значения: см. свойство `PrimaryUnitsPrecision`.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRotated 0acfa8e4>
(vla-get-AltTolerancePrecision obj)
;;; 3
(vla-put-AltTolerancePrecision obj acDimPrecisionOne)
```

8.2.33. Свойство *AltToleranceSuppressLeadingZeros*

Объекты: **DimAligned**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**.

Описание: подавление ведущих нулей в допусках альтернативных единиц примитива. Переопределяет значение, установленное системной переменной DIMALTTZ.

Возможные значения:

- `:vlax-true` — подавлять;
- `:vlax-false` — не подавлять.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRotated 0acfa8e4>
(vla-get-AltToleranceSuppressLeadingZeros obj)
;;; :vlax-true
(vla-put-AltToleranceSuppressLeadingZeros obj :vlax-false)
```

8.2.34. Свойство *AltToleranceSuppressTrailingZeros*

Объекты: **DimAligned**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**.

Описание: подавление хвостовых нулей в допусках альтернативных единиц примитива. Переопределяет значение, установленное системной переменной DIMALTTZ. Возможные значения:

- `:vlax-true` — подавлять;
- `:vlax-false` — не подавлять.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRotated 0acfa8e4>
(vla-get-AltToleranceSuppressTrailingZeros obj)
;;; :vlax-true
(vla-put-AltToleranceSuppressTrailingZeros obj :vlax-false)
```

8.2.35. Свойство *AltToleranceSuppressZeroFeet*

Объекты: **DimAligned**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**.

Описание: подавление "футовых" нулей в допусках альтернативных единиц примитива. Переопределяет значение, установленное системной переменной DIMALTTZ. Возможные значения:

- `:vlax-true` — подавлять;
- `:vlax-false` — не подавлять.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRotated 0acfa8e4>
(vla-get-AltToleranceSuppressZeroFeet obj)
;;; :vlax-true
(vla-put-AltToleranceSuppressZeroFeet obj :vlax-false)
```

8.2.36. Свойство *AltToleranceSuppressZeroInches*

Объекты: **DimAligned**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**.

Описание: подавление "дюймовых" нулей в допусках альтернативных единиц примитива. Переопределяет значение, установленное системной переменной DIMALTTZ. Возможные значения:

- `:vlax-true` — подавлять;
- `:vlax-false` — не подавлять.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRotated 0acfa8e4>
(vla-get-AltToleranceSuppressZeroInches obj)
;;; :vlax-true
(vla-put-AltToleranceSuppressZeroInches obj :vlax-false)
```

8.2.37. Свойство *AltUnits*

Объекты: **DimAligned**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**.

Описание: отображение альтернативных единиц примитива. Переопределяет значение, установленное системной переменной DIMALT. Возможные значения:

- `:vlax-true` — отображать;
- `:vlax-false` — скрывать.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRotated 0acfa8e4>
(vla-get-AltUnits obj)
;;; :vlax-false
(vla-put-AltUnits obj :vlax-true)
```

8.2.38. Свойство *AltUnitsFormat*

Объекты: **DimAligned**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**.

Описание: формат альтернативных единиц примитива. Переопределяет значение, установленное системной переменной DIMALTU. Возможные значения:

- `acDimScientific` (1) — научные;
- `acDimDecimal` (2) — десятичные;
- `acDimEngineering` (3) — технические;
- `acDimArchitecturalStacked` (4) — архитектурные (с горизонтальной чертой);

- `acDimFractionalStacked` (5) — дробные (с горизонтальной чертой);
- `acDimArchitectural` (6) — архитектурные;
- `acDimFractional` (7) — дробные;
- `acDimWindowsDesktop` (8) — по региональным настройкам операционной системы.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRotated 0acfa8e4>
(vla-get-AltUnitsFormat obj)
;;; 2
(vla-put-AltUnitsFormat obj acDimArchitecturalStacked)
```

8.2.39. Свойство *AltUnitsPrecision*

Объекты: **DimAligned**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**.

Описание: точность альтернативных единиц примитива. Переопределяет значение, установленное системной переменной DIMALTD. Возможные значения: см. свойство `PrimaryUnitsPrecision`.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRotated 0acfa8e4>
(vla-get-AltUnitsPrecision obj)
;;; 3
(vla-put-AltUnitsPrecision obj acDimPrecisionOne)
```

8.2.40. Свойство *AltUnitsScale*

Объекты: **DimAligned**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**.

Описание: масштабный коэффициент альтернативных единиц примитива. Переопределяет значение, установленное системной переменной DIMALTF. Возможные значения: положительное вещественное число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRotated 0acfa8e4>
(vla-get-AltUnitsScale obj)
;;; 0.0393701
(vla-put-AltUnitsScale obj 0.049)
```

8.2.41. Свойство *Angle*

Объект: **Line**.

Описание: угол наклона отрезка относительно оси X **в радианах**. Только чтение. Возможные значения: от нуля до 2π.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadLine 0acfd0e4>
(vla-get-Angle obj)
;;; 0.449719
```

8.2.42. Свойство *AngleFormat*

Объекты: **Dim3PointAngular**, **DimAngular**.

Описание: формат угловых единиц примитива. Переопределяет значение, установленное системной переменной DIMAUNIT. Возможные значения:

- acDegrees (0) — градусы с десятичной дробной частью;
- acDegreeMinuteSeconds (1) — градусы, минуты, секунды;
- acGrads (2) — грады;
- acRadians (3) — радианы.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimAngular 0aceba44>
(vla-get-AngleFormat obj)
;;; 0
(vla-put-AngleFormat obj acDegreeMinuteSeconds)
```

8.2.43. Свойство *AngleVertex*

Объект: **Dim3PointAngular**.

Описание: координаты точки вершины угла. Возможные значения: координаты трехмерной точки в виде варианта, содержащего безопасный массив.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDim3PointAngular 0acf2cd4>
(setq temp (vla-get-AngleVertex obj))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (419.191 101.63 0.0)
(vla-put-AngleVertex obj (vlax-3d-point (getpoint)))
```

8.2.44. Свойство *Annotation*

Объект: **Leader**.

Описание: указатель на объект, служащий аннотацией для выноски. В качестве аннотации может выступать объект типа Tolerance, MText или BlockRef. Возможные значения: указатель на объект или nil.

Примечание

Установка возможна, только если никакой объект не связан с данной выноской. С одной аннотацией могут быть связаны несколько выносок.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadLeader 0acf4234>
(setq temp (vla-get-Annotation obj))
;;; #<VLA-OBJECT IAcadMText2 0acf40f4>
(vla-Delete temp)
(vla-Update obj)
(setq new (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadBlockReference 0acfe7d4>
(vla-put-Annotation obj new)
```

8.2.45. Свойство *Application*

Объекты: **все объекты**.

Описание: указатель на приложение. Только чтение. Возможные значения: VLA-объект.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadLine 0ad1cbc4>
(vla-get-Application obj)
;;; #<VLA-OBJECT IAcadApplication 00b9d604>
```

8.2.46. Свойство *ArcEndParam*

Объект: **DimArcLength**.

Описание: значение параметра в конечной точке измеряемой дуги. Возможные значения: вещественное число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimArcLength 08fe4b74>
(vla-get-ArcEndParam obj)
;;; 2.70979
(vla-put-ArcEndParam obj (/ pi 2))
```

8.2.47. Свойство *ArcLength*

Объект: **Arc**.

Описание: длина дуги. Только чтение. Возможные значения: положительное вещественное число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadArc 0acbc1e4>
(vla-get-ArcLength obj)
;;; 234.595
```

8.2.48. Свойство *ArcPoint*

Объект: **DimArcLength**.

Описание: точка, определяющая положение размерной линии. Возможные значения: координаты трехмерной точки в виде варианта с безопасным массивом.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimArcLength 08fe4b74>
(setq temp (vla-get-ArcPoint obj))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (217.246 251.471 0.0)
(vla-put-ArcPoint obj (vlax-3d-point (getpoint)))
```

8.2.49. Свойство *ArcSmoothness*

Объекты: **PViewport**, **Viewport**.

Описание: параметр управления аппроксимацией криволинейных сегментов на экране. Возможные значения: целое число в диапазоне от 1 до 20 000.

```
(setq obj (vla-get-ActiveViewport actdoc))
;;; #<VLA-OBJECT IAcadViewport 0acd9044>
(vla-get-ArcSmoothness obj)
;;; 100
(vla-put-ArcSmoothness obj 2000)
```

8.2.50. Свойство *ArcStartParam*

Объект: **DimArcLength**.

Описание: значение параметра в начальной точке измеряемой дуги. Возможные значения: вещественное число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimArcLength 08fe4b74>
(vla-get-ArcStartParam obj)
;;; 0.857404
(vla-put-ArcStartParam obj (/ pi 2))
```

8.2.51. Свойство *Area*

Объекты: **Arc**, **Circle**, **Ellipse**, **Hatch**, **LightweightPolyline**, **Polyline**, **Region**, **Spline**.

Описание: площадь. Возможные значения: положительное вещественное число.

Примечание

Установка возможна только для `Circle`, с автоматическим изменением радиуса. Для `Hatch` свойство доступно с версии AutoCAD 2006.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadCircle 0acd9434>
(vla-get-Area obj)
;;; 3595.29
(vla-put-Area obj 4500)
```

8.2.52. Свойство *Arrowhead1Block*

Объекты: **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimDiametric**, **DimRotated**.

Описание: имя пользовательского блока для первой стрелки размерной линии примитива. Блок должен быть определен. Переопределяет значение, установленное системной переменной DIMBLK1. Возможные значения: строка с именем блока или пустая строка.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimDiametric 0acdcd84>
(vla-get-Arrowhead1Block obj)
;;; ""
(vla-put-Arrowhead1Block obj "MyArrow1")
```

8.2.53. Свойство *Arrowhead1Type*

Объекты: **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimDiametric**, **DimRotated**.

Описание: тип первой стрелки размерной линии примитива. Переопределяет значение, установленное системной переменной DIMBLK1. Возможные значения:

- `acArrowDefault` (0) — заполненная замкнутая стрелка (по умолчанию);
- `acArrowClosedBlank` (1) — пустая замкнутая стрелка;
- `acArrowClosed` (2) — замкнутая стрелка;
- `acArrowDot` (3) — точка;
- `acArrowArchTick` (4) — двойная засечка;

- acArrowOblique (5) — засечка;
- acArrowOpen (6) — разомкнутая стрелка;
- acArrowOrigin (7) — указатель начала;
- acArrowOrigin2 (8) — указатель начала-2;
- acArrowOpen90 (9) — разомкнутая стрелка с прямым углом;
- acArrowOpen30 (10) — разомкнутая стрелка с углом 30°;
- acArrowDotSmall (11) — точка малая;
- acArrowDotBlank (12) — точка контурная;
- acArrowSmall (13) — контурная малая точка;
- acArrowBoxBlank (14) — прямоугольник;
- acArrowBoxFilled (15) — заполненный прямоугольник;
- acArrowDatumBlank (16) — треугольник;
- acArrowDatumFilled (17) — заполненный треугольник;
- acArrowIntegral (18) — интеграл;
- acArrowNone (19) — нет стрелки;
- acArrowUserDefined (20) — пользовательский блок. Только чтение.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimDiametric 0acdcd84>
(vla-get-Arrowhead1Type obj)
;;; 20
(vla-put-Arrowhead1Type obj acArrowClosed)
```

8.2.54. Свойство *Arrowhead2Block*

Объекты: **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimDiametric**, **DimRotated**.

Описание: имя пользовательского блока для второй стрелки размерной линии примитива. Блок должен быть определен. Переопределяет значение, установленное системной переменной DIMBLK2. Возможные значения: строка с именем блока или пустая строка.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimDiametric 0acdcd84>
(vla-get-Arrowhead2Block obj)
;;; ""
(vla-put-Arrowhead2Block obj "MyArrow2")
```

8.2.55. Свойство *Arrowhead2Type*

Объекты: **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimDiametric**, **DimRotated**.

Описание: тип второй стрелки размерной линии примитива. Переопределяет значение, установленное системной переменной DIMBLK2. Возможные значения: см. `Arrowhead1Type`.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimDiametric 0acdcd84>
(vla-get-Arrowhead2Type obj)
;;; 20
(vla-put-Arrowhead2Type obj acArrowClosed)
```

8.2.56. Свойство *ArrowheadBlock*

Объекты: **DimRadial**, **DimRadialLarge**, **Leader**.

Описание: имя пользовательского блока для стрелки размерной линии примитива. Блок должен быть определен. Переопределяет значение, установленное системной переменной DIMLDRBLK. Возможные значения: строка с именем блока или пустая строка.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRadial 0acb87f4>
(vla-get-ArrowheadBlock obj)
;;; ""
(vla-put-ArrowheadBlock obj "MyArrow1")
```

8.2.57. Свойство *ArrowheadSize*

Объекты: **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**, **Leader**.

Описание: размер стрелки размерной линии примитива. Переопределяет значение, установленное системной переменной DIMASZ. Возможные значения: неотрицательное вещественное число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRadial 0acb87f4>
(vla-get-ArrowheadSize obj)
;;; 2.5
(vla-put-ArrowheadSize obj 1.5)
```

8.2.58. Свойство *ArrowheadType*

Объекты: **DimRadial**, **DimRadialLarge**, **Leader**.

Описание: тип стрелки размерной линии примитива. Переопределяет значение, установленное системной переменной DIMLDRBLK. Возможные значения: см. Arrowhead1Type.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRadial 0acb87f4>
(vla-get-ArrowheadType obj)
;;; 20
(vla-put-ArrowheadType obj acArrowClosed)
```

8.2.59. Свойство *AssociativeHatch*

Объект: **Hatch**.

Описание: свойство ассоциативности штриховки. Возможные значения:

- :vlax-true — ассоциативная (только чтение);
- :vlax-false — неассоциативная.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadHatch 0aceb084>
(vla-get-AssociativeHatch obj)
;;; :vlax-true
(vla-put-AssociativeHatch obj :vlax-false)
```

8.2.60. Свойство *AttachmentPoint*

Объект: **MText**.

Описание: тип точки выравнивания примитива. Возможные значения:

- acAttachmentPointTopLeft (1) — верхняя левая;
- acAttachmentPointTopCenter (2) — верхняя центральная;
- acAttachmentPointTopRight (3) — верхняя правая;
- acAttachmentPointMiddleLeft (4) — средняя левая;
- acAttachmentPointMiddleCenter (5) — средняя центральная;
- acAttachmentPointMiddleRight (6) — средняя правая;
- acAttachmentPointBottomLeft (7) — нижняя левая;
- acAttachmentPointBottomCenter (8) — нижняя центральная;
- acAttachmentPointBottomRight (9) — нижняя правая.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadMText2 0acc8504>
(vla-get-AttachmentPoint obj)
;;; 1
(vla-put-AttachmentPoint obj acAttachmentPointMiddleCenter)
```

8.2.61. Свойство *Author*

Объект: **SummaryInfo**.

Описание: имя автора документа. Возможные значения: строка (может быть пустой).

```
(setq obj (vla-get-SummaryInfo actdoc))
;;; #<VLA-OBJECT IAcadSummaryInfo 0acbc8b4>
(vla-get-Author obj)
;;; ""
(vla-put-Author obj "Пупкин В.В.")
```

8.2.62. Свойство *AutoAudit*

Объект: **PreferencesOpenSave**.

Описание: автоматическая проверка чертежа после каждого использования команд: DXFIN (ИМПОРТА), DXBIN (ИМПОРТД). Возможные значения:

- `:vlax-true` — режим проверки включен;
- `:vlax-false` — режим проверки выключен.

```
(setq obj (vla-get-OpenSave
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesOpenSave 0acc9124>
(vla-get-AutoAudit obj)
;;; :vlax-false
(vla-put-AutoAudit obj :vlax-true)
```

8.2.63. Свойство *AutomaticPlotLog* 2005

Объект: **PreferencesOutput**.

Описание: автоматическая запись в файл протокола операций печати и публикации документа. Возможные значения:

- `:vlax-true` — вести протокол;
- `:vlax-false` — не вести протокол.

```
(setq obj (vla-get-Output
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesOutput2 0accaf38>
(vla-get-AutomaticPlotLog obj)
;;; :vlax-true
(vla-put-AutomaticPlotLog obj :vlax-false)
```

8.2.64. Свойство *AutoSaveInterval*

Объект: **PreferencesOpenSave**.

Описание: интервал автосохранения документов в минутах. Соответствует системной переменной SAVETIME. Возможные значения: целое число от 0 до 600. Установка нулевого значения выключает режим автосохранения.

```
(setq obj (vla-get-OpenSave
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesOpenSave 0acc9124>
(vla-get-AutoSaveInterval obj)
;;; 10
(vla-put-AutoSaveInterval obj 45)
```

8.2.65. Свойство *AutoSavePath*

Объект: **PreferencesFiles**.

Описание: путь автосохранения документов. Соответствует системной переменной SAVEFILEPATH. Возможные значения: строка, полный путь.

Примечание

Устанавливаемый путь должен существовать.

```
(setq obj (vla-get-Files
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesFiles2 0acca860>
(vla-get-AutoSavePath obj)
;;; "C:\\Temp\\Autosave\\"
(vla-put-AutoSavePath obj "C:\\Autosave\\")
```

8.2.66. Свойство *AutoSnapAperture*

Объект: **PreferencesDrafting**.

Описание: отображение прицела объектной привязки. Соответствует системной переменной APBOX. Возможные значения:

- `:vlax-true` — отображать;
- `:vlax-false` — не отображать.

```
(setq obj (vla-get-Drafting
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesDrafting 0ad6a498>
(vla-get-AutoSnapAperture obj)
;;; :vlax-false
(vla-put-AutoSnapAperture obj :vlax-true)
```

8.2.67. Свойство *AutoSnapApertureSize*

Объект: **PreferencesDrafting**.

Описание: размер прицела объектной привязки, в пикселах. Соответствует системной переменной APERTURE. Возможные значения: целое число от 1 до 50.

```
(setq obj (vla-get-Drafting
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesDrafting 0ad6a498>
(vla-get-AutoSnapApertureSize obj)
;;; 10
(vla-put-AutoSnapApertureSize obj 5)
```

8.2.68. Свойство *AutoSnapMagnet*

Объект: **PreferencesDrafting**.

Описание: режим автопривязки. Соответствует системной переменной AUTOSNAP. Возможные значения:

- `:vlax-true` — включен;
- `:vlax-false` — выключен.

```
(setq obj (vla-get-Drafting
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesDrafting 0ad6a498>
(vla-get-AutoSnapMagnet obj)
;;; :vlax-false
(vla-put-AutoSnapMagnet obj :vlax-true)
```

8.2.69. Свойство *AutoSnapMarker*

Объект: **PreferencesDrafting**.

Описание: отображение маркера автопривязки. Соответствует системной переменной AUTOSNAP. Возможные значения:

- `:vlax-true` — отображать;
- `:vlax-false` — не отображать.

```
(setq obj (vla-get-Drafting
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesDrafting 0ad6a498>
(vla-get-AutoSnapMarker obj)
;;; :vlax-true
(vla-put-AutoSnapMarker obj :vlax-false)
```

8.2.70. Свойство *AutoSnapMarkerColor*

Объект: **PreferencesDrafting**.

Описание: цвет маркера автопривязки. Возможные значения:

- acRed (1) — красный;
- acYellow (2) — желтый;
- acGreen (3) — зеленый;
- acCyan (4) — голубой;
- acBlue (5) — синий;
- acMagenta (6) — фиолетовый;
- acWhite (7) — белый.

```
(setq obj (vla-get-Drafting
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesDrafting 0ad6a498>
(vla-get-AutoSnapMarkerColor obj)
;;; 7
(vla-put-AutoSnapMarkerColor obj acYellow)
```

8.2.71. Свойство *AutoSnapMarkerSize*

Объект: **PreferencesDrafting**.

Описание: размер маркера автопривязки. Возможные значения: целое число от 1 до 20.

```
(setq obj (vla-get-Drafting
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesDrafting 0ad6a498>
(vla-get-AutoSnapMarkerSize obj)
;;; 5
(vla-put-AutoSnapMarkerSize obj 4)
```

8.2.72. Свойство *AutoSnapToolTip*

Объект: **PreferencesDrafting**.

Описание: отображение текстовой подсказки об активном режиме привязки около курсора. Соответствует системной переменной AUTOSNAP. Возможные значения:

- :vlax-true — отображать;
- :vlax-false — не отображать.

```
(setq obj (vla-get-Drafting
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesDrafting 0ad6a498>
(vla-get-AutoSnapToolTip obj)
;;; :vlax-true
(vla-put-AutoSnapToolTip obj :vlax-false)
```

8.2.73. Свойство *AutoTrackingVecColor*

Объект: **PreferencesDisplay**.

Описание: цвет маркера объектной привязки. Возможные значения: вариант типа OLE_COLOR.

Примечание

Этот тип данных не поддерживается в AutoLISP. Тем не менее управлять цветом можно, зная цифровые коды цветов, но нельзя получить значение фактически установленного цвета.

```
(setq obj (vla-get-Display
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesDisplay 0ad6d3ac>
(setq temp (vla-get-AutoTrackingVecColor obj))
;;; #<variant 19 8421504>
(vlax-variant-value temp)
;;; error: LISPFromPtr failed. The type is not supported: 19
(vla-put-AutoTrackingVecColor obj (vlax-make-variant 12634758 19))
```

8.2.74. Свойство *AutoTrackTooltip*

Объект: **PreferencesDrafting**.

Описание: отображение текстовой подсказки около курсора об активном режиме полярного или объектного отслеживания. Соответствует системной переменной AUTOSNAP.

Возможные значения:

- `:vlax-true` — отображать;
- `:vlax-false` — не отображать.

```
(setq obj (vla-get-Drafting
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesDrafting 0ad6bd48>
(vla-get-AutoTrackTooltip obj)
;;; :vlax-true
(vla-put-AutoTrackTooltip obj :vlax-false)
```

8.2.75. Свойство *BackgroundFill*

Объект: **MText**.

Описание: использование цветного фона под примитивом. Возможные значения:

- `:vlax-true` — включено;
- `:vlax-false` — выключено.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadMText2 0ad99184>
(vla-get-BackgroundFill obj)
;;; :vlax-false
(vla-put-BackgroundFill obj :vlax-true)
```

8.2.76. Свойство *Backward*

Объекты: **Attribute**, **AttributeReference**, **Text**.

Описание: генерация текста "задом наперед". Возможные значения:

- `:vlax-true` — включена;
- `:vlax-false` — выключена.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadText2 0ad91d64>
(vla-get-Backward obj)
;;; :vlax-false
(vla-put-Backward obj :vlax-true)
```

8.2.77. Свойство *BasePoint*

Объекты: **Ray**, **XLine**.

Описание: координаты базовой точки. Возможные значения: вариант, содержащий безопасный массив с тремя координатами.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadRay 0ad95534>
(setq temp (vla-get-BasePoint obj))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (109.727 98.2878 0.0)
(vla-put-BasePoint obj (vlax-3d-point (getpoint)))
```

8.2.78. Свойство *BatchPlotProgress*

Объект: **Plot**.

Описание: индикация процесса пакетной печати. Возможные значения:

- `:vlax-true` — включена;
- `:vlax-false` — выключена.

```
(setq obj (vla-get-Plot actdoc))
;;; #<VLA-OBJECT IAcadPlot 0ad9544c>
(vla-get-BatchPlotProgress obj)
;;; :vlax-false
(vla-put-BatchPlotProgress obj :vlax-true)
```

8.2.79. Свойство *BeepOnError*

Объект: **PreferencesSystem**.

Описание: звуковое предупреждение об ошибке. Возможные значения:

- `:vlax-true` — включено;
- `:vlax-false` — выключено.

```
(setq obj (vla-get-System
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesSystem 0ad94d04>
(vla-get-BeepOnError obj)
;;; :vlax-false
(vla-put-BeepOnError obj :vlax-true)
```

8.2.80. Свойство *BigFontFile*

Объект: **TextStyle**.

Описание: имя большого шрифта (например, для отображения азиатских иероглифов) в текстовом стиле. Возможные значения: строка с именем файла без пути, но с расширением, или пустая строка. Только чтение.

Примечание

Применимо только для стилей, использующих SHX-шрифт. Установка в качестве имени пустой строки не допускается. Наличие устанавливаемого файла шрифта не проверяется.

```
(setq obj (vla-Item (vla-get-TextStyles actdoc) "Standard"))
;;; #<VLA-OBJECT IAcadTextStyle 0a9bce94>
(vla-get-BigFontFile obj)
;;; ""
(vla-put-BigFontFile obj "bigfont.shx")
```

8.2.81. Свойство *BitFlags*

2005

Объект: **TableStyle**.

Описание: битовый код. Не документирован. Возможные значения: неотрицательное целое число — 0, 1, 2, 4, 8, 9 (только чтение), 10 (только чтение). Возможно, допускаются и другие.

Примечание

Соответствует значению DXF-группы с кодом 71 стилей таблиц.

```
(setq obj (vla-Item
        (vla-Item (vla-get-dictionaries actdoc) "acad_tablestyle")
        "Standard"))
;;; #<VLA-OBJECT IAcadTableStyle 0a9ac934>
(vla-get-BitFlags obj)
;;; 0
(vla-put-BitFlags obj 8)
```

8.2.82. Свойство *Block*

Объект: **Layout**.

Описание: указатель на блок, ассоциированный с вкладкой, в том числе и **Model** (Модель). Только чтение. Возможные значения: VLA-объект.

```
(setq obj (vla-get-ActiveLayout actdoc))
;;; #<VLA-OBJECT IAcadLayout 0fa7c014>
(vla-get-Block obj)
;;; #<VLA-OBJECT IAcadModelSpace2 0fa79ae4>
```

8.2.83. Свойство *Blocks*

Объект: **Document**.

Описание: указатель на семейство блоков документа. Только чтение. Возможные значения: VLA-объект.

```
(vla-get-Blocks actdoc)
;;; #<VLA-OBJECT IAcadBlocks 0fa7aa34>
```

8.2.84. Свойство *BlockScaling*

Объект: **Block**.

Описание: режим масштабирования по осям. Возможные значения:

- acAny (0) — любые масштабные коэффициенты;
- acUniform (1) — только равные.

```
(setq obj (vla-Item (vla-get-Blocks actdoc) 3))
;;; #<VLA-OBJECT IAcadBlock3 08ff56e4>
(vla-get-BlockScaling obj)
;;; 1
(vla-put-BlockScaling obj 0)
```

8.2.85. Свойство *Blue*

Объект: **AcCmColor**.

Описание: значение синей цветовой компоненты. Только чтение. Возможные значения: целое число в диапазоне 0—255.

```
(setq obj (vla-GetInterfaceObject
            (vlax-get-acad-object) "AutoCAD.AcCmColor.16"))
;;; #<VLA-OBJECT IAcadAcCmColor 0fa7c550>
(vla-get-Blue obj)
;;; 0
```

8.2.86. Свойство *BookName*

Объект: **AcCmColor**.

Описание: имя цветового альбома. Только чтение. Возможные значения: строка с именем альбома или пустая строка.

```
(setq obj (vla-GetInterfaceObject
            (vlax-get-acad-object) "AutoCAD.AcCmColor.16"))
;;; #<VLA-OBJECT IAcadAcCmColor 0fa7c550>
(vla-get-BookName obj)
;;; ""
```

8.2.87. Свойство *Brightness*

Объект: **Raster**.

Описание: яркость растра. Возможные значения: целое число в диапазоне 0—100.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadRasterImage 0faaa814>
(vla-get-Brightness obj)
;;; 50
(vla-put-Brightness obj 30)
```

8.2.88. Свойство *CanonicalMediaName*

Объекты: **Layout**, **PlotConfiguration**.

Описание: каноническое имя формата бумаги. Возможные значения: строка.

Примечание

Написание канонического имени может не совпадать с тем именем, которое выбирается пользователем, например: при выборе пользовательского формата в окне настройки печати, в качестве канонического имени будет нечто вроде "User256". Именно каноническое имя используется в настройке печати программным способом с применением ActiveX.

```
(setq obj (vla-get-ActiveLayout actdoc))
;;; #<VLA-OBJECT IAcadLayout 0a9a16d4>
(vla-get-CanonicalMediaName obj)
;;; "A4"
(vla-put-CanonicalMediaName obj "A3")
```

8.2.89. Свойство *Caption*

Объекты: **Application**, **PopupMenuItem**.

Описание: текст заголовка. Только чтение. Возможные значения: строка.

```
(setq obj (vla-Item (vla-Item (vla-get-Menus
            (vla-Item (vla-get-MenuGroups (vlax-get-acad-object))
                      "acad")) 9) 8))
;;; #<VLA-OBJECT IAcadPopupMenuItem 7ef2f0a8>
(vla-get-Caption obj)
;;; "Cop&y"
```

8.2.90. Свойство *Catalog*

Объекты: **LabelTemplate**, **LinkTemplate**, **Query**.

Описание: имя каталога, содержащего таблицу. Возможные значения: строка.

```
(setq DBConn (vla-GetInterfaceObject (vlax-get-acad-object)
               "CAO.DbConnect.16"))
;;; #<VLA-OBJECT IDbConnect 056de7d0>
```

```
(setq DBLinksT (vlax-invoke DBConn 'GetLinkTemplates))
;;; #<VLA-OBJECT ILinkTemplates 056cc4f0>
(setq DBLinkT (vla-Item DBLinksT 0)) ;_был создан заранее
;;; #<VLA-OBJECT ILinkTemplate 0570ee20>
(vlax-get-property DBLinkT 'Catalog)
;;; ""
(vlax-put-property DBLinkT 'Catalog "Any string")
```

8.2.91. Свойство *CategoryName*

2005

Объект: **View**.

Описание: категория. Возможные значения: строка с именем или пустая строка.

```
(setq obj (vla-Item (vla-get-Views actdoc) 0))
;;; #<VLA-OBJECT IAcadView2 115da294>
(vla-get-CategoryName obj)
;;; ""
(vla-put-CategoryName obj "Presentation")
```

8.2.92. Свойство *Center*

Объекты: **Arc**, **Circle**, **DimRadialLarge**, **Ellipse**, **PViewport**, **View**, **Viewport**.

Описание: координаты центра объекта. Возможные значения: вариант, содержащий безопасный массив с тремя координатами.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadCircle 0a85c614>
(setq temp (vla-get-Center obj))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (159.903 156.059 0.0)
(vla-put-Center obj (vlax-3d-point (getpoint)))
```

8.2.93. Свойство *CenterMarkSize*

Объекты: **DimDiametric**, **DimRadial**, **DimRadialLarge**.

Описание: размер маркера центра. Переопределяет значение, установленное системной переменной DIMCEN. Возможные значения: неотрицательное вещественное число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimDiametric 10eca314>
(vla-get-CenterMarkSize obj)
;;; 2.5
(vla-put-CenterMarkSize obj 4.0)
```

8.2.94. Свойство *CenterPlot*

Объекты: **Layout**, **PlotConfiguration**.

Описание: центрирование области печати на листе бумаги. Возможные значения:

- `:vlax-true` — включено;
- `:vlax-false` — выключено.

Примечание

Установка невозможна в случае, если область печати — `Layout`. Для актуализации изменений в рисунке требуется регенерация.

```
(setq obj (vla-get-ActiveLayout actdoc))
;;; #<VLA-OBJECT IAcadLayout 0a87c364>
(vla-get-CenterPlot obj)
;;; :vlax-false
(vla-put-CenterPlot obj :vlax-true)
```

8.2.95. Свойство *CenterPoint*

Объект: **DimArcLength**.

Описание: точка центра. Возможные значения: координаты трехмерной точки в виде варианта с безопасным массивом.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimArcLength 09006eb4>
(setq temp (vla-get-CenterPoint obj))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (260.244 41.4932 0.0)
(vla-put-CenterPoint obj (vlax-3d-point (getpoint)))
```

8.2.96. Свойство *CenterType*

Объекты: **DimDiametric**, **DimRadial**, **DimRadialLarge**.

Описание: тип маркера центра. Переопределяет значение, установленное системной переменной DIMCEN. Возможные значения:

- `acCenterMark` (0) — маркер;
- `acCenterLine` (1) — линии;
- `acCenterNone` (2) — нет.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimAligned 0ae19584>
```

```
(vla-get-CenterType obj)
;;; 0
(vla-put-CenterType obj acCenterLine)
```

8.2.97. Свойство *Centroid*

Объекты: **3DSolid**, **Region**.

Описание: координаты центроида. Только чтение. Возможные значения: вариант, содержащий координаты трехмерной (двумерной — для области) точки в виде варианта типа безопасного массива.

Примечание

Получение центроида области возможно, только если он лежит в плоскости *XY* текущей системы координат.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadRegion 0a88d584>
(setq temp (vla-get-Centroid obj))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (98.0333 105.672)
```

8.2.98. Свойство *Check*

Объект: **PopupMenuItem**.

Описание: отметка (флажок) на пункте меню. Возможные значения:

- `:vlax-true` — включена;
- `:vlax-false` — выключена.

Примечание

Изменение значения этого свойства пункта меню не означает изменение макроса данного пункта. В приведенном ниже примере отметка устанавливается на пункте **Copy** (Копировать) падающего меню **Modify** (Редакт). Номера меню и пунктов соответствуют версии AutoCAD 2005.

```
(setq obj (vla-Item (vla-Item (vla-get-Menus
            (vla-Item (vla-get-MenuGroups (vlax-get-acad-object))
                      "acad")) 9) 8))
;;; #<VLA-OBJECT IAcadPopupMenuItem 7ef2f0a8>
(vla-get-Check obj)
;;; :vlax-false
(vla-put-Check obj :vlax-true)
```

8.2.99. Свойство *ChordPoint*

2006

Объект: **DimRadialLarge**.

Описание: точка хорды размерной линии. Возможные значения: координаты трехмерной точки в виде варианта с безопасным массивом.

Примечание

Поскольку изменение данного свойства приводит к неконтролируемому искажению геометрии объекта, рекомендуется использование только для чтения.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRadialLarge 0ea18114>
(setq temp (vla-get-ChordPoint obj))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (287.859 175.419 0.0)
(vla-put-ChordPoint obj (vlax-3d-point (getpoint)))
```

8.2.100. Свойство *Circumference*

Объект: **Circle**.

Описание: длина окружности. При установке автоматически изменяет радиус. Возможные значения: положительное вещественное число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadCircle 0a8986b4>
(vla-get-Circumference obj)
;;; 209.814
(vla-put-Circumference obj 300)
```

8.2.101. Свойство *Clipped*

Объект: **PViewport**.

Описание: наличие подрезки. Только чтение. Возможные значения:

- `:vlax-true` — есть;
- `:vlax-false` — нет.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadPViewport2 0a8a1414>
(vla-get-Clipped obj)
;;; :vlax-false
```

8.2.102. Свойство *ClippingEnabled*

Объект: **Raster**.

Описание: включение подрезки. Возможные значения:

- `:vlax-true` — включена;
- `:vlax-false` — выключена.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadRasterImage 0a8abc84>
(vla-get-ClippingEnabled obj)
;;; :vlax-true
(vla-put-ClippingEnabled obj :vlax-false)
```

8.2.103. Свойство *Closed*

Объекты: **3DPolyline**, **LightweightPolyline**, **Polyline**, **Spline**.

Описание: замкнутость контура (`Spline` — только чтение). Возможные значения:

- `:vlax-true` — замкнут;
- `:vlax-false` — не замкнут.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadLWPolyline 0a88b364>
(vla-get-Closed obj)
;;; :vlax-false
(vla-put-Closed obj :vlax-true)
```

8.2.104. Свойство *Color*

Объекты: **все графические объекты**, **Group**, **Layer**, **LabelTemplate**.

Описание: цвет (`Group` — только чтение). Возможные значения: целое число от 0 до 256, в том числе:

- `acByBlock` (0) — по блоку (кроме `LabelTemplate`);
- `acRed` (1) — красный;
- `acYellow` (2) — желтый;
- `acGreen` (3) — зеленый;
- `acCyan` (4) — голубой;
- `acBlue` (5) — синий;
- `acMagenta` (6) — фиолетовый;
- `acWhite` (7) — белый;
- `acByLayer` (256) — по слою (кроме `Layer` и `LabelTemplate`).

Примечание

О доступе к объекту `LabelTemplate` см. свойство `Catalog`.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadLWPolyline 0a88b364>
(vla-get-Color obj)
;;; 256
(vla-put-Color obj acRed)
```

8.2.105. Свойство *ColorBookPath*

Объект: **`PreferencesFiles`**.

Описание: путь к файлу альбома цветов. Возможные значения: строка. Не проверяет существование пути.

```
(setq obj (vla-get-Files
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesFiles2 00fe3c40>
(vla-get-ColorBookPath obj)
;;; "C:\\Program Files ... \\color;C:\\Documents and ... \\color"
(vla-put-ColorBookPath obj "E:\\New Color Path")
```

8.2.106. Свойство *ColorIndex*

Объект: **`AcCmColor`**.

Описание: номер цвета в палитре. Возможные значения: целое число в диапазоне 0—256. *См. также свойство* `Color`.

```
(setq obj (vla-GetInterfaceObject
            (vlax-get-acad-object) "AutoCAD.AcCmColor.16"))
;;; #<VLA-OBJECT IAcadAcCmColor 0a8a6120>
(vla-get-ColorIndex obj)
;;; 256
(vla-put-ColorIndex obj 2)
```

8.2.107. Свойство *ColorMethod*

Объект: **`AcCmColor`**.

Описание: метод задания цвета. Возможные значения:

- `acColorMethodByLayer` (192) — по слою;
- `acColorMethodByBlock` (193) — по блоку;
- `acColorMethodByRGB` (194) — RGB;

- `acColorMethodByACI` (195) — по индексу;
- `acColorMethodForeground` (197) — приоритетный.

```
(setq obj (vla-GetInterfaceObject
            (vlax-get-acad-object) "AutoCAD.AcCmColor.16"))
;;; #<VLA-OBJECT IAcadAcCmColor 0a8a6120>
(vla-get-ColorMethod obj)
;;; 192
(vla-put-ColorMethod obj acColorMethodByACI)
```

8.2.108. Свойство *ColorName*

Объект: **`AcCmColor`**.

Описание: имя цвета. Только чтение. Возможные значения: строка с именем цвета или пустая строка.

```
(setq obj (vla-GetInterfaceObject
            (vlax-get-acad-object) "AutoCAD.AcCmColor.16"))
;;; #<VLA-OBJECT IAcadAcCmColor 0a8a6120>
(vla-get-ColorName obj)
;;; ""
```

8.2.109. Свойство *Columns*

Объекты: **`MInsertBlock`**, **`Table`**.

Описание: количество столбцов массива блоков (установка только для `MInsertBlock`). Возможные значения: положительное целое число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadMInsertBlock 0ac09d34>
(vla-get-Columns obj)
;;; 4
(vla-put-Columns obj 2)
```

8.2.110. Свойство *ColumnSpacing*

Объект: **`MInsertBlock`**.

Описание: расстояние между столбцами. Возможные значения: вещественное число, включая 0 и отрицательные.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadMInsertBlock 0ac09d34>
(vla-get-ColumnSpacing obj)
;;; 59.2912
(vla-put-ColumnSpacing obj 52)
```

8.2.111. Свойство *ColumnWidth*

2005

Объект: **Table**.

Описание: ширина всех столбцов. Только установка. Возможные значения: положительное вещественное число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0ac08084>
(vla-put-ColumnWidth obj 40)
```

8.2.112. Свойство *Comment*

Объект: **SecurityParams**.

Описание: комментарий, включаемый в цифровую подпись рисунка. Возможные значения: строка, в том числе пустая.

```
(setq obj (vla-GetInterfaceObject
            (vlax-get-acad-object) "AutoCAD.SecurityParams.16"))
;;; #<VLA-OBJECT IAcadSecurityParams 0ad4b0f4>
(vla-get-Comment obj)
;;; ""
(vla-put-Comment obj "Vasia Pupkin CopyLeft")
```

8.2.113. Свойство *Comments*

Объект: **SummaryInfo**.

Описание: комментарий в свойствах рисунка. Возможные значения: строка, в том числе пустая.

```
(setq obj (vla-get-SummaryInfo actdoc))
;;; #<VLA-OBJECT IAcadSummaryInfo 0ad4cb94>
(vla-get-Comments obj)
;;; ""
(vla-put-Comments obj "Any comment for my drawing")
```

8.2.114. Свойство *ConfigFile*

Объект: **PreferencesFiles**.

Описание: полное имя файла конфигурации. Только чтение. Возможные значения: строка с именем файла.

```
(setq obj (vla-get-Files
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesFiles2 0ad4c890>
```

```
(vla-get-ConfigFile obj)
;;; "C:\\Documents ... enu\\acad2005.cfg"
```

8.2.115. Свойство *ConfigName*

Объекты: **Layout**, **PlotConfiguration**.

Описание: имя конфигурации драйвера печати. Возможные значения: строка.

```
(setq obj (vla-get-ActiveLayout actdoc))
;;; #<VLA-OBJECT IAcadLayout 0ad4c764>
(vla-get-ConfigName obj)
;;; "Default Windows System Printer.pc3"
(vla-put-ConfigName obj "PublishToWeb DWF55.pc3")
```

8.2.116. Свойство *Constant*

Объекты: **Attribute**, **AttributeReference**.

Описание: признак постоянности атрибута (установка только для `Attribute`). Возможные значения:

- `:vlax-true` — включен;
- `:vlax-false` — выключен.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadAttribute 0ad71f04>
(vla-get-Constant obj)
;;; :vlax-false
(vla-put-Constant obj :vlax-true)
```

8.2.117. Свойство *ConstantWidth*

Объекты: **LightweightPolyline**, **Polyline**.

Описание: постоянная ширина. Возможные значения: неотрицательное вещественное число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadLWPolyline 0ad68284>
(vla-get-ConstantWidth obj)
;;; 0.0
(vla-put-ConstantWidth obj 45)
```

8.2.118. Свойство *ContinuousPlotLog*

Объект: **PreferencesOutput**.

Описание: ведение единого протокола печати.

Возможные значения:

- `:vlax-true` — включено;
- `:vlax-false` — выключено.

```
(setq obj (vla-get-Output
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesOutput2 0ad704f8>
(vla-get-ContinuousPlotLog obj)
;;; :vlax-true
(vla-put-ContinuousPlotLog obj :vlax-false)
```

8.2.119. Свойство *ContourLinesPerSurface*

Объект: **DatabasePreferences**.

Описание: число образующих линий при отображении твердотельных примитивов на экране. Соответствует системной переменной ISOLINES. Возможные значения: целое число от 0 до 2047.

```
(setq obj (vla-get-Preferences actdoc))
;;; #<VLA-OBJECT IAcadDatabasePreferences 0a97cf54>
(vla-get-ContourLinesPerSurface obj)
;;; 4
(vla-put-ContourLinesPerSurface obj 15)
```

8.2.120. Свойство *Contrast*

Объект: **Raster**.

Описание: контрастность растра. Возможные значения: целое число от 0 до 100.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadRasterImage 0a98e184>
(vla-get-Contrast obj)
;;; 50
(vla-put-Contrast obj 70)
```

8.2.121. Свойство *ControlPoints*

Объект: **Spline**.

Описание: координаты управляющих точек. Возможные значения: вариант (безопасный массив), содержащий трехмерные координаты в виде плоского списка.

Примечание

Координаты точек в списке представлены в виде (X_1 Y_1 Z_1 ... X_N Y_N Z_N), количество управляющих точек на 2 больше, чем вершин.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadSpline 0a99f154>
(setq temp (vla-get-ControlPoints obj))
;;; #<variant 8197 ...>
(setq temp (vlax-safearray->list (vlax-variant-value temp)))
;;; (383.65 295.88 0.0 359.142 322.974 0.0 301.766 386.402 0.0 394.306 390.276
0.0 447.317 392.496 0.0)
(setq temp (subst 410.0 (nth 7 temp) temp))
;;; Заменяем значение Y третьей управляющей точки.
;;; (383.65 295.88 0.0 359.142 322.974 0.0 301.766 410.0 0.0 394.306 390.276
0.0 447.317 392.496 0.0)
(vla-put-ControlPoints obj (vlax-safearray-fill
    (vlax-make-safearray vlax-vbDouble '(0 . 14)) temp))
```

8.2.122. Свойство *Coordinate*

Объекты: **3DPoly**, **Leader**, **LightweightPolyline**, **Point**, **PolyfaceMesh**, **PolygonMesh**, **Polyline**, **Solid**, **Trace**.

Описание: координаты точки по порядковому номеру (отсчет начинается с нуля). Возможные значения: вариант (безопасный массив), содержащий координаты одной точки; размерность зависит от типа примитива.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadLWPolyline 0a99d294>
(setq temp (vla-get-Coordinate obj 2))
;;; Выбираем координаты третьей точки.
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (566.39 424.428)
(vla-put-Coordinate obj 2 (vlax-safearray-fill
    (vlax-make-safearray vlax-vbDouble '(0 . 1)) '(200.0 400.0)))
;;; Изменяем координаты третьей точки.
```

8.2.123. Свойство *Coordinates*

Объекты: **3DFace**, **3DPolyline**, **Leader**, **LightweightPolyline**, **MLine**, **Point**, **PolyfaceMesh**, **PolygonMesh**, **Polyline**, **Solid**, **Trace**.

Описание: координаты всех точек. Возможные значения: вариант (безопасный массив), содержащий трехмерные координаты в виде плоского списка.

Примечание

Координаты точек в списке представлены в виде (X_1 Y_1 Z_1 ... X_N Y_N Z_N).

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcad3DPolyline 0a999944>
```

```
(setq temp (vla-get-Coordinates obj))
;;; #<variant 8197 ...>
(setq temp (vlax-safearray->list (vlax-variant-value temp)))
;;; (392.398 490.749 0.0 437.111 530.05 0.0 458.982 462.091 0.0 425.933
452.266 0.0)
(setq temp (subst 550.0 (nth 3 temp) temp))
;;; Заменяем значение X второй точки.
;;; (392.398 490.749 0.0 550.0 530.05 0.0 458.982 462.091 0.0 425.933 452.266
0.0)
(vla-put-Coordinates obj (vlax-safearray-fill
    (vlax-make-safearray vlax-vbDouble '(0 . 11)) temp))
```

8.2.124. Свойство *Count*

Объекты: **все семейства**, **Block**, **Dictionary**, **Group**, **SelectionSet**.

Описание: количество объектов, содержащихся в семействе. Возможные значения: целое неотрицательное число.

```
(setq obj (vla-get-Layouts actdoc))
;;; #<VLA-OBJECT IAcadLayouts 0a99ffb4>
(vla-get-Count obj)
;;; 3
```

8.2.125. Свойство *CreateBackup*

Объект: **PreferencesOpenSave**.

Описание: создание резервной копии (BAK-файла) при сохранении документа. Соответствует системной переменной ISAVEBAK. Возможные значения:

- `:vlax-true` — включено;
- `:vlax-false` — выключено.

```
(setq obj (vla-get-OpenSave
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesOpenSave 0a99e664>
(vla-get-CreateBackup obj)
;;; :vlax-true
(vla-put-CreateBackup obj :vlax-false)
```

8.2.126. Свойство *CursorSize*

Объект: **PreferencesDisplay**.

Описание: размер перекрестия курсора в процентах от размера экрана. Соответствует системной переменной CURSORSIZE. Возможные значения: целое число от 1 до 100.

```
(setq obj (vla-get-Display
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesDisplay 0a99e53c>
(vla-get-CursorSize obj)
;;; 5
(vla-put-CursorSize obj 10)
```

8.2.127. Свойство *CustomDictionary*

Объект: **PreferencesFiles**.

Описание: пользовательский словарь. Возможные значения: строка с именем файла. Не проверяет наличие устанавливаемого файла.

```
(setq obj (vla-get-Files
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesFiles2 00feec20>
(vla-get-CustomDictionary obj)
;;; "C:\\Documents ... enu\\support\\sample.cus"
(vla-put-CustomDictionary obj "C:\\Temp\\mydict.cus")
```

8.2.128. Свойство *CustomScale*

Объект: **PViewport**.

Описание: пользовательский масштаб. Возможные значения: неотрицательное вещественное число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadPViewport2 0a9b36f4>
(vla-get-CustomScale obj)
;;; 0.40927
(vla-put-CustomScale obj 0.5)
```

8.2.129. Свойство *Database*

Объект: **Document**.

Описание: указатель на базу документа. Только чтение. Возможные значения: VLA-объект.

```
(vla-get-Database actdoc)
;;; #<VLA-OBJECT IAcadDatabase 0b10b594>
```

8.2.130. Свойство *DataSource*

Объекты: **LabelTemplate**, **LinkTemplate**, **Query**.

Описание: имя источника данных. Возможные значения: строка.

Примечание

Свойство доступно для изменения только при отсутствии связей, использующих шаблон, с которым производятся манипуляции. При установке значение не проверяется, но для функционирования совместимость источника данных обязательна.

```
(setq DBConn (vla-GetInterfaceObject (vlax-get-acad-object)
               "CAO.DbConnect.16"))
;;; #<VLA-OBJECT IDbConnect 056de7d0>
(setq DBLinksT (vlax-invoke DBConn 'GetLinkTemplates))
;;; #<VLA-OBJECT ILinkTemplates 056cc4f0>
(setq DBLinkT (vla-Item DBLinksT 0)) ;_был создан заранее
;;; #<VLA-OBJECT ILinkTemplate 0570ee20>
(vlax-get-property DBLinkT 'DataSource)
;;; "jet_dbsamples"
(vlax-put-property DBLinkT 'DataSource "Tutor Base")
```

8.2.131. Свойство *DataSourceLocation*

Объект: **DbConnect**.

Описание: расположение, по умолчанию, файла источника данных. Только чтение. Возможные значения: строка.

```
(setq DBConn (vla-GetInterfaceObject (vlax-get-acad-object)
               "CAO.DbConnect.16"))
;;; #<VLA-OBJECT IDbConnect 056de7d0>
(vlax-get-property DBConn 'DataSourceLocation)
;;; "C:\\Doc ... enu\\Data Links"
```

8.2.132. Свойство *DbConnect*

Объекты: **LabelTemplate**, **LabelTemplates**, **Link**, **Links**, **LinkTemplate**, **LinkTemplates**, **Queries**, **Query**.

Описание: получение корневого объекта. Только чтение. Возможные значения: VLA-объект.

```
(setq obj (vlax-invoke (vla-GetInterfaceObject (vlax-get-acad-object)
    "CAO.DbConnect.16") 'GetLinkTemplates))
;;; #<VLA-OBJECT ILinkTemplates 057567d0>
(vlax-get-property obj 'DbConnect)
;;; #<VLA-OBJECT IDbConnect 05756790>
```

8.2.133. Свойство *DecimalSeparator*

Объекты: **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**, **Tolerance**.

Описание: формат десятичного разделителя для размерного текста. Переопределяет значение, установленное системной переменной DIMDSEP. Возможные значения: строка, содержащая один литерал.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRotated 0b0f32e4>
(vla-get-DecimalSeparator obj)
;;; ","
(vla-put-DecimalSeparator obj ".")
```

8.2.134. Свойство *DefaultInternetURL*

Объект: **PreferencesFiles**.

Описание: адрес сайта (URL), открываемого командой BROWSER (БРАУЗЕР) по умолчанию. Возможные значения: строка, допускается пустая.

```
(setq obj (vla-get-Files
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesFiles2 0b110d60>
(vla-get-DefaultInternetURL obj)
;;; "http://www.autodesk.com"
(vla-put-DefaultInternetURL obj "http://www.uniip.ru")
```

8.2.135. Свойство *DefaultOutputDevice*

Объект: **PreferencesOutput**.

Описание: устройство, назначаемое вкладкам для печати или публикации документа по умолчанию. Возможные значения: строка с именем драйвера, значение проверяется.

```
(setq obj (vla-get-Output
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesOutput2 0b110d18>
(vla-get-DefaultOutputDevice obj)
;;; "Microsoft Office Document Image Writer"
(vla-put-DefaultOutputDevice obj "Autodesk DWF Writer")
```

8.2.136. Свойство *DefaultPlotStyleForLayer*

Объект: **PreferencesOutput**.

Описание: стиль печати для слоев по умолчанию. Соответствует системной переменной DEFLPLSTYLE. Возможные значения: строка с именем стиля, значение проверяется.

```
(setq obj (vla-get-Output
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesOutput2 0b110d18>
(vla-get-DefaultPlotStyleForLayer obj)
;;; "Normal"
(vla-put-DefaultPlotStyleForLayer obj "Style 1")
```

8.2.137. Свойство *DefaultPlotStyleForObjects*

Объект: **PreferencesOutput**.

Описание: стиль печати для примитивов по умолчанию. Соответствует системной переменной DEFPLSTYLE. Возможные значения: строка с именем стиля, значение проверяется.

```
(setq obj (vla-get-Output
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesOutput2 0b110d18>
(vla-get-DefaultPlotStyleForObjects obj)
;;; "Normal"
(vla-put-DefaultPlotStyleForObjects obj "ByLayer")
```

8.2.138. Свойство *DefaultPlotStyleTable*

Объект: **PreferencesOutput**.

Описание: файл таблицы стилей печати по умолчанию для новых документов. Возможные значения: строка с именем файла или пустая строка, значение проверяется.

Примечание

Устанавливаемая таблица стилей должна соответствовать типу печати.

```
(setq obj (vla-get-Output
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesOutput2 0b110d18>
(vla-get-DefaultPlotStyleTable obj)
;;; ""
(vla-put-DefaultPlotStyleTable obj "monochrome.stb")
```

8.2.139. Свойство *DefaultPlotToFilePath*

Объект: **PreferencesOutput**.

Описание: путь при печати в файл по умолчанию. Возможные значения: строка с путем или пустая строка, значение не проверяется.

```
(setq obj (vla-get-Output
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesOutput2 0b110d18>
(vla-get-DefaultPlotToFilePath obj)
;;; "C:\\Temp"
(vla-put-DefaultPlotToFilePath obj "C:\\PlotOutput")
```

8.2.140. Свойство *DefinedSize*

Объект: **KeyDescription**.

Описание: максимальная ширина поля данных. Возможные значения: неотрицательное целое число.

```
(setq DBConn (vla-GetInterfaceObject (vlax-get-acad-object)
               "CAO.DbConnect.16"))
;;; #<VLA-OBJECT IDbConnect 056de7d0>
(setq DBLinksT (vlax-invoke DBConn 'GetLinkTemplates))
;;; #<VLA-OBJECT ILinkTemplates 056cc4f0>
(setq DBLinkT (vla-Item DBLinksT 0)) ;_был создан заранее
;;; #<VLA-OBJECT ILinkTemplate 0570ee20>
(setq KeyDeskrs (vlax-get-property DBLinkT 'KeyDescriptions))
;;; #<VLA-OBJECT IKeyDescriptions 0546ca10>
(setq KeyDeskr (vla-Item KeyDeskrs 0))
;;; #<VLA-OBJECT IKeyDescription 0546dbd0>
(vlax-get-property KeyDeskr 'DefinedSize)
;;; 0
(vlax-put-property KeyDeskr 'DefinedSize 10)
```

8.2.141. Свойство *Degree*

Объект: **Spline**.

Описание: порядок полиномиального представления. Только чтение. Возможные значения: целое число от 1 до 25.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadSpline 0b11c4f4>
(vla-get-Degree obj)
;;; 3
```

8.2.142. Свойство *Delta*

Объект: **Line**.

Описание: разности по трем координатам (ΔX, ΔY, ΔZ). Только чтение. Возможные значения: вариант, содержащий безопасный массив из трех вещественных чисел.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadLine 0b11c8c4>
(setq temp (vla-get-Delta obj))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (-27.0438 -46.8413 0.0)
```

8.2.143. Свойство *DemandLoadARXApp*

Объект: **PreferencesOpenSave**.

Описание: загрузка ARX-приложений по требованию. Возможные значения (могут суммироваться):

- acDemandLoadDisabled (0) — отключена;
- acDemandLoadOnObjectDetect (1) — при обнаружении объекта приложения;
- acDemandLoadCmdInvoke (2) — при вызове команды.

```
(setq obj (vla-get-OpenSave
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesOpenSave 0b1186e4>
(vla-get-DemandLoadARXApp obj)
;;; 3
(vla-put-DemandLoadARXApp obj acDemandLoadOnObjectDetect)
```

8.2.144. Свойство *Description*

Объекты: **DynamicBlockReferenceProperty**, **Layer**, **Linetype**, **TableStyle**.

Описание: пояснение. Возможные значения: строка, в том числе пустая.

```
(setq obj (vla-Item (vla-get-Layers actdoc) "0"))
;;; #<VLA-OBJECT IAcadLayer2 0b11c694>
(vla-get-Description obj)
;;; ""
(vla-put-Description obj "Служебный слой")
```

8.2.145. Свойство *Diameter*

Объект: **Circle**.

Описание: диаметр. Возможные значения: положительное вещественное число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadCircle 1616b534>
(vla-get-Diameter obj)
;;; 46.9027
(vla-put-Diameter obj 100)
```

8.2.146. Свойство *Dictionaries*

Объект: **Document**.

Описание: словари. Возможные значения: указатель на VLA-объект семейства.

```
(vla-get-Dictionaries actdoc)
;;; #<VLA-OBJECT IAcadDictionaries 0acbb264>
```

8.2.147. Свойство *DimensionLineColor*

Объекты: **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimDiametric**, **DimRadial**, **DimRadialLarge**, **DimRotated**, **Leader**, **Tolerance**.

Описание: цвет размерной линии. Переопределяет значение, установленное системной переменной DIMCLRD. Возможные значения: целое число от 0 до 256, в том числе:

- `acByBlock` (0) — по блоку;
- `acRed` (1) — красный;
- `acYellow` (2) — желтый;
- `acGreen` (3) — зеленый;
- `acCyan` (4) — голубой;
- `acBlue` (5) — синий;
- `acMagenta` (6) — фиолетовый;
- `acWhite` (7) — белый;
- `acByLayer` (256) — по слою.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimAligned 0acbd264>
(vla-get-DimensionLineColor obj)
;;; 0
(vla-put-DimensionLineColor obj acYellow)
```

8.2.148. Свойство *DimensionLineExtend*

Объекты: **DimAligned**, **DimArcLength**, **DimRotated**.

Описание: выступ размерной линии за выносные линии. Переопределяет значение, установленное системной переменной DIMDLE. Возможные значения: неотрицательное вещественное число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimAligned 0acbd264>
(vla-get-DimensionLineExtend obj)
;;; 0.0
(vla-put-DimensionLineExtend obj 1.75)
```

8.2.149. Свойство *DimensionLineWeight*

Объекты: **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**.

Описание: вес для размерной линии. Переопределяет значение, установленное системной переменной DIMLWD. Возможные значения:

- `acLnWtByLayer` (–1) — ByLayer (ПоСлою);
- `acLnWtByBlock` (–2) — ByBlock (ПоБлоку);
- `acLnWtByLwDefault` (–3) — Default (Обычный);
- `acLnWt000` (0) — 0 мм;
- `acLnWt005` (5) — 0.05 мм;
- `acLnWt009` (9) — 0.09 мм;
- `acLnWt013` (13) — 0.13 мм;
- `acLnWt015` (15) — 0.15 мм;
- `acLnWt018` (18) — 0.18 мм;
- `acLnWt020` (20) — 0.2 мм;
- `acLnWt025` (25) — 0.25 мм;
- `acLnWt030` (30) — 0.3 мм;
- `acLnWt035` (35) — 0.35 мм;
- `acLnWt040` (40) — 0.4 мм;
- `acLnWt050` (50) — 0.5 мм;
- `acLnWt053` (53) — 0.53 мм;
- `acLnWt060` (60) — 0.6 мм;
- `acLnWt070` (70) — 0.7 мм;
- `acLnWt080` (80) — 0.8 мм;
- `acLnWt090` (90) — 0.9 мм;
- `acLnWt100` (100) — 1 мм;
- `acLnWt106` (106) — 1.06 мм;
- `acLnWt120` (120) — 1.2 мм;
- `acLnWt140` (140) — 1.4 мм;
- `acLnWt158` (158) — 1.58 мм;
- `acLnWt200` (200) — 2 мм;
- `acLnWt211` (211) — 2.11 мм.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimAligned 0acbd264>
(vla-get-DimensionLineWeight obj)
;;; -2
(vla-put-DimensionLineWeight obj acLnWt020)
```

8.2.150. Свойство *DimLine1Suppress*

Объекты: **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimDiametric**, **DimRotated**.

Описание: подавление начальной части размерной линии. Переопределяет значение, установленное системной переменной DIMSD1. Возможные значения:

- `:vlax-true` — включено;
- `:vlax-false` — выключено.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimAligned 0acbd264>
(vla-get-DimLine1Suppress obj)
;;; :vlax-false
(vla-put-DimLine1Suppress obj :vlax-true)
```

8.2.151. Свойство *DimLine2Suppress*

Объекты: **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimDiametric**, **DimRotated**.

Описание: подавление конечной части размерной линии. Переопределяет значение, установленное системной переменной DIMSD2. Возможные значения:

- `:vlax-true` — включено;
- `:vlax-false` — выключено.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimAligned 0acbd264>
(vla-get-DimLine2Suppress obj)
;;; :vlax-false
(vla-put-DimLine2Suppress obj :vlax-true)
```

8.2.152. Свойство *DimLineInside*

Объекты: **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimRotated**.

Описание: отображение размерной линии только между выносными линиями. Переопределяет значение, установленное системной переменной DIMSOXD.

Возможные значения:

- `:vlax-true` — включено;
- `:vlax-false` — выключено.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimAligned 0acbd264>
(vla-get-DimLineInside obj)
;;; :vlax-false
(vla-put-DimLineInside obj :vlax-true)
```

8.2.153. Свойство *DimLineSuppress*

Объекты: **DimRadial**, **DimRadialLarge**.

Описание: подавление размерной линии. Переопределяет значение, установленное системной переменной DIMSD2. Возможные значения:

- `:vlax-true` — включено;
- `:vlax-false` — выключено.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRadial 0acc2f74>
(vla-get-DimLineSuppress obj)
;;; :vlax-false
(vla-put-DimLineSuppress obj :vlax-true)
```

8.2.154. Свойство *DimStyles*

Объект: **Document**.

Описание: семейство размерных стилей. Только чтение. Возможные значения: указатель на VLA-объект семейства.

```
(vla-get-DimStyles actdoc)
;;; #<VLA-OBJECT IAcadDimStyles 0accf4e4>
```

8.2.155. Свойство *Direction*

Объекты: **Table**, **View**, **Viewport**, **PViewport**.

Описание: вектор направления взгляда. Возможные значения: вариант, содержащий безопасный массив с тремя координатами.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadPViewport2 0ac755f4>
(setq temp (vla-get-Direction obj))
;;; #<variant 8197 ...>
```

```
(vlax-safearray->list (vlax-variant-value temp))
;;; (90.7872 -117.867 159.67)
(vla-put-Direction obj (vlax-3d-point '(90.0 -60.0 60.0)))
```

8.2.156. Свойство *DirectionVector*

Объекты: **Ray**, **Tolerance**, **XLine**.

Описание: вектор направления. Возможные значения: вариант, содержащий безопасный массив с тремя координатами.

Примечание

В свойствах примитива вектор направления хранится в виде координат мнимой точки на поверхности единичной сферы, при этом устанавливать можно произвольные координаты, которые будут спроецированы на единичную сферу автоматически.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadRay 0ae2c114>
(setq temp (vla-get-DirectionVector obj))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (0.696511 0.717546 0.0)
(vla-put-DirectionVector obj (vlax-3d-point '(1.0 1.0 0.0)))
;;; Посмотрим, что хранит свойство:
(setq temp (vla-get-DirectionVector obj))
(vlax-safearray->list (vlax-variant-value temp))
;;; (0.707107 0.707107 0.0)
;;; А теперь трехкоординатный вариант:
(vla-put-DirectionVector obj (vlax-3d-point '(1.0 1.0 1.0)))
(setq temp (vla-get-DirectionVector obj))
(vlax-safearray->list (vlax-variant-value temp))
;;; (0.57735 0.57735 0.57735)
```

8.2.157. Свойство *Display*

Объект: **Preferences**.

Описание: указатель на объект, содержащий экранные настройки. Только чтение. Возможные значения: указатель на VLA-объект.

```
(setq obj (vla-get-Preferences (vlax-get-acad-object)))
;;; #<VLA-OBJECT IAcadPreferences 0ae2d3fc>
(vla-get-Display obj)
;;; #<VLA-OBJECT IAcadPreferencesDisplay 0ae2d3dc>
```

8.2.158. Свойство *DisplayableAttributes*

Объект: **LabelTemplate**.

Описание: имена колонок, которые используются в шаблоне ярлыка. Только чтение. Возможные значения: безопасный массив строк в виде варианта.

Примечание

Об изменении отображаемых атрибутов см. свойство `Format`.

```
(setq DBConn (vla-GetInterfaceObject (vlax-get-acad-object)
               "CAO.DbConnect.16"))
;;; #<VLA-OBJECT IDbConnect 04de13f0>
(setq DBLablsT (vlax-invoke DBConn 'GetLabelTemplates))
;;; #<VLA-OBJECT ILabelTemplates 04de0460>
(setq DBLablT (vla-Item DBLablsT 0)) ;_был создан заранее
;;; #<VLA-OBJECT ILabelTemplate 04de1090>
(setq tmp (vlax-get-property DBLablT 'DisplayableAttributes))
;;; #<variant 8200 ...>
(vlax-safearray->list (vlax-variant-value tmp))
;;; ("Manufacturer")
```

8.2.159. Свойство *DisplayGrips*

Объект: **PreferencesSelection**.

Описание: отображение ручек. Соответствует системной переменной GRIPS. Возможные значения:

- `:vlax-true` — включено;
- `:vlax-false` — выключено.

```
(setq obj (vla-get-Selection
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesSelection 0ae2d170>
(vla-get-DisplayGrips obj)
;;; :vlax-true
(vla-put-DisplayGrips obj :vlax-false)
```

8.2.160. Свойство *DisplayGripsWithinBlocks*

Объект: **PreferencesSelection**.

Описание: отображение ручек внутри блоков. Соответствует системной переменной GRIPBLOCK. Возможные значения:

- `:vlax-true` — включено;
- `:vlax-false` — выключено.

```
(setq obj (vla-get-Drafting
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesSelection 0ae2d170>
(vla-get-DisplayGripsWithinBlocks obj)
;;; :vlax-false
(vla-put-DisplayGripsWithinBlocks obj :vlax-true)
```

8.2.161. Свойство *DisplayLayoutTabs*

Объект: **PreferencesDisplay**.

Описание: отображение ярлыков вкладок. Возможные значения:

- `:vlax-true` — включено;
- `:vlax-false` — выключено.

```
(setq obj (vla-get-Display
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesDisplay 0ae25c0c>
(vla-get-DisplayLayoutTabs obj)
;;; :vlax-true
(vla-put-DisplayLayoutTabs obj :vlax-false)
```

8.2.162. Свойство *DisplayLocked*

Объект: **PViewport**.

Описание: блокировка. Возможные значения:

- `:vlax-true` — включена;
- `:vlax-false` — выключена.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadPViewport2 0ae4e4a4>
(vla-get-DisplayLocked obj)
;;; :vlax-false
(vla-put-DisplayLocked obj :vlax-true)
```

8.2.163. Свойство *DisplayOLEScale*

Объект: **PreferencesSystem**.

Описание: показ диалога масштабирования при вставке OLE-объекта. Возможные значения:

- `:vlax-true` — включен;
- `:vlax-false` — выключен.

```
(setq obj (vla-get-System
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesSystem 0ae4f7e4>
(vla-get-DisplayOLEScale obj)
;;; :vlax-true
(vla-put-DisplayOLEScale obj :vlax-false)
```

8.2.164. Свойство *DisplayScreenMenu*

Объект: **PreferencesDisplay**.

Описание: отображение экранного меню. Возможные значения:

- `:vlax-true` — включено;
- `:vlax-false` — выключено.

```
(setq obj (vla-get-Display
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesDisplay 0ae25c0c>
(vla-get-DisplayScreenMenu obj)
;;; :vlax-false
(vla-put-DisplayScreenMenu obj :vlax-true)
```

8.2.165. Свойство *DisplayScrollBars*

Объект: **PreferencesDisplay**.

Описание: отображение полос прокрутки. Возможные значения:

- `:vlax-true` — включено;
- `:vlax-false` — выключено.

```
(setq obj (vla-get-Display
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesDisplay 0ae25c0c>
(vla-get-DisplayScrollBars obj)
;;; :vlax-false
(vla-put-DisplayScrollBars obj :vlax-true)
```

8.2.166. Свойство *DisplaySilhouette*

Объект: **DatabasePreferences**.

Описание: показ только силуэта объемных примитивов при каркасном отображении. Соответствует системной переменной DISPSILH. Возможные значения:

- `:vlax-true` — включен;
- `:vlax-false` — выключен.

```
(vla-get-DisplaySilhouette (vla-get-Preferences actdoc))
;;; :vlax-false
(vla-put-DisplaySilhouette
  (vla-get-Preferences actdoc) :vlax-true)
```

8.2.167. Свойство *DisplayValues*

Объект: **Label**.

Описание: отображаемые значения ярлыка. Только чтение. Возможные значения: вариант (безопасный массив), содержащий строки в виде варианта.

```
(setq DBConn (vla-GetInterfaceObject (vlax-get-acad-object)
             "CAO.DbConnect.16"))
;;; #<VLA-OBJECT IDbConnect 04de13f0>
(setq DBLinksT (vlax-invoke DBConn 'GetLinkTemplates))
;;; #<VLA-OBJECT ILinkTemplates 04df12d0>
(setq DBLinkT (vla-Item DBLinksT "Link2"))
;;; #<VLA-OBJECT ILinkTemplate 04dff0d0>
(setq Links (vlax-invoke-method DBConn 'GetLinks DBLinkT
  (vlax-make-variant (vlax-safearray-fill (vlax-make-safearray
    vlax-vbLong '(0 . 0)) (list (vla-get-ObjectID (vlax-ename->vla-object
      (car (entsel))))))) 2 actdoc))
;;; #<VLA-OBJECT ILinks 0566baf0>
(setq Link (vla-Item Links 1))
;;; #<VLA-OBJECT ILink 0566fd30>
(setq obj (vla-get-Label Link))
;;; #<VLA-OBJECT ILabel 0566d620>
(setq tmp (vlax-get-property obj 'DisplayValues))
;;; #<variant 8204 ...>
(setq tmp (vlax-safearray->list (vlax-variant-value tmp)))
;;; (#<variant 8 SUN>)
(vlax-variant-value (car tmp))
;;; "SUN"
```

8.2.168. Свойство *DockedVisibleLines*

Объект: **PreferencesDisplay**.

Описание: число видимых строк в зоне командной строки. Возможные значения: положительное целое число.

Примечание

Программная установка этого свойства игнорируется.

```
(setq obj (vla-get-Display
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesDisplay 0b13bf2c>
```

```
(vla-get-DockedVisibleLines obj)
;;; 3
(vla-put-DockedVisibleLines obj 5)
```

8.2.169. Свойство *DockStatus*

Объект: **Toolbar**.

Описание: позиция на экране. Только чтение. Возможные значения:

- 0 (acToolbarDockTop) — верхняя фиксация;
- 1 (acToolbarDockBottom) — нижняя фиксация;
- 2 (acToolbarDockLeft) — левая фиксация;
- 3 (acToolbarDockRight) — правая фиксация;
- 4 (acToolbarFloating) — плавающая.

```
(setq obj (vla-Item (vla-get-Toolbars (vla-Item
  (vla-get-MenuGroups (vlax-get-acad-object)) "acad")) "Layers"))
;;; #<VLA-OBJECT IAcadToolbar 0101223c>
(vla-get-DockStatus obj)
;;; 0
```

8.2.170. Свойство *Document*

Объекты: **все графические объекты**, **Block**, **Blocks**, **Dictionaries**, **Dictionary**, **DimStyle**, **DimStyles**, **Group**, **Groups**, **LabelTemplate**, **LabelTemplates**, **Layer**, **Layers**, **Layout**, **Layouts**, **Linetype**, **Linetypes**, **Link**, **Links**, **LinkTemplate**, **LinkTemplates**, **ModelSpace**, **PaperSpace**, **PlotConfiguration**, **PlotConfigurations**, **Queries**, **Query**, **RegisteredApplication**, **RegisteredApplications**, **TextStyle**, **TextStyles**, **UCS**, **UCSs**, **View**, **Viewport**, **Viewports**, **Views**, **XRecord**.

Описание: указатель на документ. Только чтение. Возможные значения: VLA-объект (документ).

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadLine 0b17aca4>
(vla-get-Document obj)
;;; #<VLA-OBJECT IAcadDocument 01045f90>
```

8.2.171. Свойство *Documents*

Объект: **Application**.

Описание: указатель на семейство документов. Только чтение. Возможные значения: VLA-объект (семейство).

```
(setq obj (vlax-get-acad-object))
;;; #<VLA-OBJECT IAcadApplication 00b9d604>
(vla-get-Documents obj)
;;; #<VLA-OBJECT IAcadDocuments 0b179150>
```

8.2.172. Свойство *Drafting*

Объект: **Preferences**.

Описание: указатель на объект настроек черчения. Только чтение. Возможные значения: VLA-объект.

```
(setq obj (vla-get-Preferences (vlax-get-acad-object)))
;;; #<VLA-OBJECT IAcadPreferences 0b17912c>
(vla-get-Drafting obj)
;;; #<VLA-OBJECT IAcadPreferencesDrafting 0b179108>
```

8.2.173. Свойство *DrawingDirection*

Объект: **MText**.

Описание: направление текста. Возможные значения:

- 1 (acLeftToRight) — слева направо;
- 2 (acRightToLeft) — справа налево (зарезервировано на будущее);
- 3 (acTopToBottom) — сверху вниз;
- 4 (acBottomToTop) — снизу вверх (зарезервировано на будущее);
- 5 (acByStyle) — по стилю.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadMText2 0b188214>
(vla-get-DrawingDirection obj)
;;; 5
(vla-put-DrawingDirection obj acTopToBottom)
```

8.2.174. Свойство *DriversPath*

Объект: **PreferencesFiles**.

Описание: путь поиска драйверов. Возможные значения: строка с путем. Не проверяет существование пути.

Примечание

Стандартный путь поиска драйверов сохраняется в любом случае.

```
(setq obj (vla-get-Files
            (vla-get-Preferences (vlax-get-acad-object))))
(setq temp (vla-get-DriversPath obj))
;;; "C:\\Program Files\\Autodesk ... \\drv"
(vla-put-DriversPath obj "E:\\New Drv Path")
(vla-get-DriversPath obj)
;;; "E:\\New Drv Path;C:\\Program Files\\Autodesk ... \\drv"
(vla-put-DriversPath obj temp)
(vla-get-DriversPath obj)
;;; "C:\\Program Files\\Autodesk ... \\drv"
```

8.2.175. Свойство *EffectiveName*

2006

Объекты: **BlockRef**, **MInsertBlock**.

Описание: оригинальное имя блока. Только чтение. Возможные значения: строка.

Примечание

Оригинальное имя блока может отличаться от того, которое показывается пользователю, что возможно при использовании динамических блоков.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadBlockReference2 090e3a54>
(vla-get-EffectiveName obj)
;;; "testdbl"
```

8.2.176. Свойство *Elevation*

Объекты: **Hatch**, **LightWeightPolyline**, **Polyline**.

Описание: уровень. Возможные значения: вещественное число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadHatch 0b195bb4>
(vla-get-Elevation obj)
;;; 0.0
(vla-put-Elevation obj 300)
```

8.2.177. Свойство *ElevationModelSpace*

Объект: **Document**.

Описание: текущий уровень пространства модели. Возможные значения: вещественное число.

```
(vla-get-ElevationModelSpace actdoc)
;;; 0.0
(vla-put-ElevationModelSpace actdoc 200)
```

8.2.178. Свойство *ElevationPaperSpace*

Объект: **Document**.

Описание: текущий уровень пространства листа. Возможные значения: вещественное число.

```
(vla-get-ElevationPaperSpace actdoc)
;;; 0.0
(vla-put-ElevationPaperSpace actdoc 100)
```

8.2.179. Свойство *Enable*

Объект: **PopupMenuItem**.

Описание: доступность пункта меню. Возможные значения:

- `:vlax-true` — доступен;
- `:vlax-false` — не доступен.

Примечание

Изменение значения этого свойства пункта меню не означает блокировку или разблокировку команды, которая вызывается этим пунктом. В приведенном ниже примере блокируется пункт **Copy** (Копировать) падающего меню **Modify** (Редакт), что не мешает пользоваться этой командой из командной строки или из панели.

```
(setq obj (vla-Item (vla-Item (vla-get-Menus
            (vla-Item (vla-get-MenuGroups (vlax-get-acad-object))
            "acad")) 9) 8))
;;; #<VLA-OBJECT IAcadPopupMenuItem 7ef2f0a8>
(vla-get-Enable obj)
;;; :vlax-true
(vla-put-Enable obj :vlax-false)
```

8.2.180. Свойство *EnableStartupDialog*

Объект: **PreferencesSystem**.

Описание: показ стартового диалога. Возможные значения:

- `:vlax-true` — включен;
- `:vlax-false` — выключен.

```
(setq obj (vla-get-System
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesSystem 0ae55354>
(vla-get-EnableStartupDialog obj)
;;; :vlax-false
(vla-put-EnableStartupDialog obj :vlax-true)
```

8.2.181. Свойство *EndAngle*

Объекты: **Arc**, **Ellipse**.

Описание: конечный угол. Возможные значения: от нуля до 2π.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadEllipse 0ae579b4>
(vla-get-EndAngle obj)
;;; 6.28319
;;; замкнутый эллипс
(vla-put-EndAngle obj (* 1.5 pi))
```

8.2.182. Свойство *EndParameter*

Объект: **Ellipse**.

Описание: конечный параметр. Возможные значения: от нуля до 2π.

Примечание

Начальный и конечный параметры вычисляются по формуле эллипса: `P(Theta) = A*cos(Theta) + B*sin(Theta)`, где `A` и `B` — большая и малая полуоси эллипса соответственно.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadEllipse 0ae579b4>
(vla-get-EndParameter obj)
;;; 6.28319
(vla-put-EndParameter obj (* 1.5 pi))
```

8.2.183. Свойство *EndPoint*

Объекты: **Arc**, **Line**, **Ellipse**.

Описание: конечная точка. Возможные значения: координаты трехмерной точки в виде варианта (безопасного массива).

Примечание

Установка только для `Line`.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadLine 0ad4b614>
(setq temp (vla-get-EndPoint obj))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (234.362 180.856 0.0)
(vla-put-EndPoint obj (vlax-3d-point (getpoint)))
```

8.2.184. Свойство *EndSubMenuLevel*

Объект: **PopupMenuItem**.

Описание: признак конечного пункта подменю. Возможные значения: неотрицательное целое число.

Примечание

Изменение значения этого свойства пункта меню подразумевает указание количества уровней меню, прерываемых данным пунктом. Фактически любая установка сохраняется и игнорируется. В приведенном ниже примере манипуляции проводятся на пункте меню **Modify | Clip | Viewport** (Редакт | Подрезка | Видовой экран).

```
(setq obj (vla-Item (vla-get-SubMenu (vla-Item (vla-Item (vla-get-Menus
(vla-Item (vla-get-MenuGroups (vlax-get-acad-object)) "acad")) 9) 3)) 2))
;;; #<VLA-OBJECT IAcadPopupMenuItem 7ef2cd28>
(vla-get-EndSubMenuLevel obj)
;;; 1
(vla-put-EndSubMenuLevel obj 0)
```

8.2.185. Свойство *EndTangent*

Объект: **Spline**.

Описание: вектор касательной конца сплайна. Возможные значения: координаты трехмерной точки в виде варианта (безопасного массива).

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadSpline 0ad3c7f4>
(setq temp (vla-get-EndTangent obj))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (0.546569 -0.837414 0.0)
(vla-put-EndTangent obj (vlax-3d-point '(1.0 1.0 0.0)))
```

8.2.186. Свойство *EntityColor*

Объект: **AcCmColor**.

Описание: цифровое значение цвета в полноцветном представлении. Возможные значения: 32-битовое целое число.

```
(setq obj (vla-GetInterfaceObject (vlax-get-acad-object)
            "AutoCAD.AcCmColor.16"))
;;; #<VLA-OBJECT IAcadAcCmColor 0ad468f0>
(vla-get-EntityColor obj)
;;; -1073741824
(vla-put-EntityColor obj -1036280892)
```

8.2.187. Свойство *Explodable*

2006

Объект: **Block**.

Описание: признак разрешения расчленения. Возможные значения:

- `:vlax-true` — разрешено;
- `:vlax-false` — запрещено.

```
(setq obj (vla-Item (vla-get-Blocks actdoc) 3))
;;; #<VLA-OBJECT IAcadBlock3 08ff56e4>
(vla-get-Explodable obj)
;;; :vlax-true
(vla-put-Explodable obj :vlax-false)
```

8.2.188. Свойство *ExtensionLineColor*

Объекты: **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimOrdinate**, **DimRotated**.

Описание: цвет выносных линий. Переопределяет значение, установленное системной переменной DIMCLRE. Возможные значения: целое число от 0 до 256, в том числе:

- `acByBlock` (0) — ByBlock (ПоБлоку);
- `acRed` (1) — красный;
- `acYellow` (2) — желтый;
- `acGreen` (3) — зеленый;
- `acCyan` (4) — голубой;
- `acBlue` (5) — синий;
- `acMagenta` (6) — фиолетовый;
- `acWhite` (7) — белый (на темном фоне) или черный (на светлом фоне);
- `acByLayer` (256) — ByLayer (ПоСлою).

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimAligned 0ad3bf84>
(vla-get-ExtensionLineColor obj)
;;; 0
(vla-put-ExtensionLineColor obj 111)
```

8.2.189. Свойство *ExtensionLineExtend*

Объекты: **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimRotated**.

Описание: величина выступа выносных линий за размерную линию. Переопределяет значение, установленное системной переменной DIMEXE. Возможные значения: неотрицательное вещественное число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimAligned 0ad3bf84>
(vla-get-ExtensionLineExtend obj)
;;; 1.25
(vla-put-ExtensionLineExtend obj 1.75)
```

8.2.190. Свойство *ExtensionLineOffset*

Объекты: **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRotated**.

Описание: величина отступа выносных линий от точки указания. Переопределяет значение, установленное системной переменной DIMEXO. Возможные значения: неотрицательное вещественное число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimAligned 0ad3bf84>
(vla-get-ExtensionLineOffset obj)
;;; 0.625
(vla-put-ExtensionLineOffset obj 0.75)
```

8.2.191. Свойство *ExtensionLineWeight*

Объекты: **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimRotated**.

Описание: вес выносных линий. Переопределяет значение, установленное системной переменной DIMLWE. Возможные значения: см. свойство `DimensionLineWeight`.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimAligned 0acbd264>
(vla-get-ExtensionLineWeight obj)
;;; -2
(vla-put-ExtensionLineWeight obj acLnWt020)
```

8.2.192. Свойство *ExtLine1EndPoint*

Объекты: **Dim3PointAngular**, **DimAngular**.

Описание: координаты конечной точки первой выносной линии. Возможные значения: координаты трехмерной точки в виде варианта (безопасного массива).

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDim3PointAngular 0ad42d84>
(setq temp (vla-get-ExtLine1EndPoint obj))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (145.058 162.424 0.0)
(vla-put-ExtLine1EndPoint obj (vlax-3d-point (getpoint)))
```

8.2.193. Свойство *ExtLine1Point*

Объект: **DimAligned**, **DimArcLength**.

Описание: координаты точки указания первой выносной линии. Возможные значения: координаты трехмерной точки в виде варианта (безопасного массива).

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimAligned 0ad3bf84>
(setq temp (vla-get-ExtLine1Point obj))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (128.983 152.822 0.0)
(vla-put-ExtLine1Point obj (vlax-3d-point (getpoint)))
```

8.2.194. Свойство *ExtLine1StartPoint*

Объект: **DimAngular**.

Описание: координаты начальной точки первой выносной линии. Возможные значения: координаты трехмерной точки в виде варианта (безопасного массива).

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimAngular 0ad39f34>
(setq temp (vla-get-ExtLine1StartPoint obj))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (159.47 166.519 0.0)
(vla-put-ExtLine1StartPoint obj (vlax-3d-point (getpoint)))
```

8.2.195. Свойство *ExtLine1Suppress*

Объекты: **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimRotated**.

Описание: подавление первой выносной линии. Переопределяет значение, установленное системной переменной DIMSE1. Возможные значения:

- `:vlax-true` — включено;
- `:vlax-false` — выключено.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDim3PointAngular 0ad42734>
(vla-get-ExtLine1Suppress obj)
;;; :vlax-false
(vla-put-ExtLine1Suppress obj :vlax-true)
```

8.2.196. Свойство *ExtLine2EndPoint*

Объекты: **Dim3PointAngular**, **DimAngular**.

Описание: координаты конечной точки второй выносной линии. Возможные значения: координаты трехмерной точки в виде варианта (безопасного массива).

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDim3PointAngular 0ad42d84>
(setq temp (vla-get-ExtLine2EndPoint obj))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (200.234 175.765 0.0)
(vla-put-ExtLine2EndPoint obj (vlax-3d-point (getpoint)))
```

8.2.197. Свойство *ExtLine2Point*

Объект: **DimAligned**, **DimArcLength**.

Описание: координаты точки указания второй выносной линии. Возможные значения: координаты трехмерной точки в виде варианта (безопасного массива).

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimAligned 0ad4a074>
(setq temp (vla-get-ExtLine2Point obj))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (259.154 233.977 0.0)
(vla-put-ExtLine2Point obj (vlax-3d-point (getpoint)))
```

8.2.198. Свойство *ExtLine2StartPoint*

Объект: **DimAngular**.

Описание: координаты начальной точки второй выносной линии. Возможные значения: координаты трехмерной точки в виде варианта (безопасного массива).

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimAngular 0ad39f34>
(setq temp (vla-get-ExtLine2StartPoint obj))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (190.643 26.1252 0.0)
(vla-put-ExtLine2StartPoint obj (vlax-3d-point (getpoint)))
```

8.2.199. Свойство *ExtLine2Suppress*

Объекты: **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimRotated**.

Описание: подавление второй выносной линии примитива. Переопределяет значение, установленное системной переменной DIMSE2. Возможные значения:

- `:vlax-true` — включено;
- `:vlax-false` — выключено.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDim3PointAngular 0ad42734>
(vla-get-ExtLine2Suppress obj)
;;; :vlax-false
(vla-put-ExtLine2Suppress obj :vlax-true)
```

8.2.200. Свойство *Fade*

Объект: **Raster**.

Описание: затухание растра (слияние с фоном), в процентах. Возможные значения: целое число от 0 до 100.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadRasterImage 0ad7abf4>
(vla-get-Fade obj)
;;; 0
(vla-put-Fade obj 60)
```

8.2.201. Свойство *Feature*

Объект: **FileDependency**.

Описание: тип внешнего файла. Возможные значения: строка вида "Acad:<тип>".

```
(setq obj (vla-Item (vla-get-FileDependencies actdoc) 1))
;;; #<VLA-OBJECT IAcadFileDependency 0ad86c74>
(vla-get-Feature obj)
;;; "Acad:Image"
```

8.2.202. Свойство *FieldLength*

Объекты: **Attribute**, **AttributeReference**.

Описание: размер поля данных. Возможные значения: неотрицательное целое число.

Примечание

Назначение неясно — любую установку AutoCAD игнорирует.

```
(setq obj (vlax-ename->vla-object (car (nentsel))))
;;; #<VLA-OBJECT IAcadAttributeReference 0ad914d4>
(vla-get-FieldLength obj)
;;; 0
(vla-put-FieldLength obj 5)
```

8.2.203. Свойство *FieldName*

Объекты: **KeyDescription**, **KeyValue**.

Описание: имя поля. Возможные значения: строка, не проверяется.

```
(setq DBConn (vla-GetInterfaceObject (vlax-get-acad-object)
               "CAO.DbConnect.16"))
;;; #<VLA-OBJECT IDbConnect 04de13f0>
(setq DBLinksT (vlax-invoke DBConn 'GetLinkTemplates))
;;; #<VLA-OBJECT ILinkTemplates 04df12d0>
(setq DBLinkT (vla-Item DBLinksT "Link2"))
;;; #<VLA-OBJECT ILinkTemplate 04dff0d0>
(setq KeysDsk (vlax-get-property DBLinkT 'KeyDescriptions))
;;; #<VLA-OBJECT IKeyDescriptions 0566a7a0>
(setq obj (vla-Item KeysDsk 0))
;;; #<VLA-OBJECT IKeyDescription 0566bb80>
(vlax-get-property obj 'FieldName)
;;; "MANUFACTURER"
(vlax-put-property obj 'FieldName "Any string")
```

8.2.204. Свойство *FileDependencies*

Объект: **Document**.

Описание: файловые зависимости. Возможные значения: VLA-объект (семейство).

```
(vla-get-FileDependencies actdoc)
;;; #<VLA-OBJECT IAcadFileDependencies 0ad7ef74>
```

8.2.205. Свойство *FileName*

Объект: **FileDependency**.

Описание: имя внешнего файла. Возможные значения: строка с именем файла.

```
(setq obj (vla-Item (vla-get-FileDependencies actdoc) 1))
;;; #<VLA-OBJECT IAcadFileDependency 0ad86c74>
(vla-get-FileName obj)
;;; "PC170048.tif"
```

8.2.206. Свойство *Files*

Объект: **Preferences**.

Описание: объект, содержащий файловые настройки. Возможные значения: указатель (VLA-объект).

```
(setq obj (vla-get-Preferences (vlax-get-acad-object)))
;;; #<VLA-OBJECT IAcadPreferences 17a3f31c>
(vla-get-Files obj)
;;; #<VLA-OBJECT IAcadPreferencesFiles2 17a3f300>
```

8.2.207. Свойство *FileSize*

Объект: **FileDependency**.

Описание: размер внешнего файла в байтах. Возможные значения: неотрицательное целое число.

```
(setq obj (vla-Item (vla-get-FileDependencies actdoc) 1))
;;; #<VLA-OBJECT IAcadFileDependency 0ad86c74>
(vla-get-FileSize obj)
;;; 23984992
```

8.2.208. Свойство *FingerprintGuid*

Объект: **FileDependency**.

Описание: GUID (глобально уникальный идентификатор интерфейса) внешнего файла. Возможные значения: строка, в том числе пустая.

```
(setq obj (vla-Item (vla-get-FileDependencies actdoc) 1))
;;; #<VLA-OBJECT IAcadFileDependency 0ad86c74>
(vla-get-FingerprintGuid obj)
;;; ""
```

8.2.209. Свойство *Fit*

Объекты: **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimDiametric**, **DimRadial**, **DimRadialLarge**, **DimRotated**.

Описание: режим вписывания элементов размера между выносными линиями. Переопределяет значение, установленное системной переменной DIMATFIT. Возможные значения:

- `acTextAndArrows` (0) — текст и стрелки;
- `acArrowsOnly` (1) — стрелки;

- `acTextOnly` (2) — текст;
- `acBestFit` (3) — что лучше впишется.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRotated 184db274>
(vla-get-Fit obj)
;;; 3
(vla-put-Fit obj acTextAndArrows)
```

8.2.210. Свойство *FitPoints*

Объект: **Spline**.

Описание: координаты определяющих точек. Возможные значения: вариант (безопасный массив), содержащий трехмерные координаты в виде плоского списка.

Примечание

Координаты точек в списке представлены в виде (X_1 Y_1 Z_1 ... X_N Y_N Z_N).

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadSpline 184e4274>
(setq temp (vla-get-FitPoints obj))
;;; #<variant 8197 ...>
(setq temp (vlax-safearray->list (vlax-variant-value temp)))
;;; (1720.95 824.922 0.0 2124.49 759.461 0.0 2018.3 554.232 0.0 2369.62
367.579 0.0)
(setq temp (subst 480.0 (nth 7 temp) temp))
;;; Заменяем значение Y третьей контрольной точки.
;;; (1720.95 824.922 0.0 2124.49 759.461 0.0 2018.3 480.0 0.0 2369.62 367.579
0.0)
(vla-put-FitPoints obj (vlax-safearray-fill
    (vlax-make-safearray vlax-vbDouble '(0 . 11)) temp))
```

8.2.211. Свойство *FitTolerance*

Объект: **Spline**.

Описание: точность (допуск) прохождения сплайна через опорные точки. Возможные значения: неотрицательное вещественное число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadSpline 184e4274>
(vla-get-FitTolerance obj)
;;; 1.0e-005
(vla-put-FitTolerance obj 50)
```

8.2.212. Свойство *FloatingRows*

Объект: **Toolbar**.

Описание: число рядов кнопок панели инструментов. Возможные значения: неотрицательное целое число, 0 приравнивается к 1.

Примечание

При установке панель должна находиться в плавающем режиме.

```
(setq obj (vla-Item (vla-get-Toolbars (vla-Item (vla-get-MenuGroups
          (vlax-get-acad-object)) "acad")) "Draw"))
;;; #<VLA-OBJECT IAcadToolbar 0100f0cc>
(vla-get-FloatingRows obj)
;;; 1
(vla-put-FloatingRows obj 3)
```

8.2.213. Свойство *FlowDirection*

Объекты: **Table**, **TableStyle**.

Описание: положение шапки (сверху или снизу). Возможные значения:

- `acTopToBottom` (0) — сверху;
- `acBottomToTop` (1) — снизу.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 184d53f4>
(vla-get-FlowDirection obj)
;;; 0
(vla-put-FlowDirection obj acBottomToTop)
```

8.2.214. Свойство *Flyout*

Объект: **ToolbarItem**.

Описание: панель инструментов, которой соответствует групповая кнопка. Только чтение. Возможные значения: VLA-объект (панель).

```
(setq obj (vla-Item (vla-Item (vla-get-Toolbars (vla-Item
(vla-get-MenuGroups (vlax-get-acad-object)) "acad")) "Draw") "Block"))
;;; #<VLA-OBJECT IAcadToolbarItem 010302bc>
(vla-get-Flyout obj)
;;; #<VLA-OBJECT IAcadToolbar 01011eec>
```

8.2.215. Свойство *FontFile*

Объект: **TextStyle**.

Описание: имя шрифта для текстового стиля. Наличие устанавливаемого файла шрифта проверяется. Возможные значения: строка (имя файла без пути).

Примечание

При установке SHX-шрифта достаточно передавать только имя без расширения, а в случае установки TTF-шрифта необходимо передавать имя **файла** шрифта (а не имя шрифта) с расширением и полным путем к файлу шрифта. Установка в качестве имени пустой строки не допускается.

```
(setq obj (vla-Item (vla-get-TextStyles actdoc) "Standard"))
;;; #<VLA-OBJECT IAcadTextStyle 101d3924>
(vla-get-FontFile obj)
;;; "txt"
(vla-put-FontFile obj "simplex") ;_ SHX-шрифт
(vla-put-FontFile obj "c:/winnt/fonts/verdana.ttf") ;_ TTF-шрифт
```

8.2.216. Свойство *FontFileMap*

Объект: **PreferencesFiles**.

Описание: файл подстановки шрифтов. Возможные значения: строка (имя файла). Проверяет наличие устанавливаемого файла.

```
(setq obj (vla-get-Files
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesFiles2 101e45e0>
(vla-get-FontFileMap obj)
;;; "C:\\Documents and ... \\enu\\support\\[FONTMAPFILENAME]"
(vla-put-FontFileMap obj "C:\\Temp\\MyMapFile.fmp")
```

8.2.217. Свойство *ForceLineInside*

Объекты: **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimDiametric**, **DimRadial**, **DimRadialLarge**, **DimRotated**.

Описание: принудительная отрисовка размерной линии между выносными. Переопределяет значение, установленное системной переменной DIMTOFL. Возможные значения:

- `:vlax-true` — включена;
- `:vlax-false` — выключена.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRotated 10202e44>
```

```
(vla-get-ForceLineInside obj)
;;; :vlax-true
(vla-put-ForceLineInside obj :vlax-false)
```

8.2.218. Свойство *Format*

Объект: **LabelTemplate**.

Описание: формат поля ярлыка. Возможные значения: форматированная строка.

Примечание

Допускается форматирование, применяемое в многострочных текстах. Отображаемые поля задаются в формате #(имя поля).

```
(setq DBConn (vla-GetInterfaceObject (vlax-get-acad-object)
               "CAO.DbConnect.16"))
;;; #<VLA-OBJECT IDbConnect 04de13f0>
(setq DBLinksT (vlax-invoke DBConn 'GetLinkTemplates))
;;; #<VLA-OBJECT ILinkTemplates 04df12d0>
(setq DBLablsT (vlax-invoke DBConn 'GetLabelTemplates))
;;; #<VLA-OBJECT ILabelTemplates 0115de40>
(setq DBLablT (vla-Item DBLablsT 0)) ;_был создан заранее
;;; #<VLA-OBJECT ILabelTemplate 011136c0>
(vlax-get-property DBLablT 'Format)
;;; "Изготовитель - #(Manufacturer)"
(vlax-put-property DBLablT 'Format
    "Изготов: #(Manufacturer)\\P{\\fVerdana;#(Equipment_Description)}")
(vlax-invoke-method DBLablT 'Reload) ;_актуализация имеющихся ярлыков
```

8.2.219. Свойство *FoundPath*

Объект: **FileDependency**.

Описание: путь внешнего файла. Возможные значения: строка с путем.

```
(setq obj (vla-Item (vla-get-FileDependencies actdoc) 1))
;;; #<VLA-OBJECT IAcadFileDependency 1827f994>
(vla-get-FoundPath obj)
;;; "C:\\Program Files\\Autodesk Building Systems 2005\\Fonts\\"
```

8.2.220. Свойство *FractionFormat*

Объекты: **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**, **Tolerance**.

Описание: формат отображения дробной части примитива. Переопределяет значение, установленное системной переменной DIMFRAC.

Возможные значения:

- `acHorizontal` (0) — дробь с горизонтальной чертой;
- `acDiagonal` (1) — дробь с диагональной чертой;
- `acNotStacked` (2) — без дроби (например, 2/3).

Примечание

Актуально только для размеров с единицами измерения `Architectural` или `Fractional`.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRotated 18280e74>
(vla-get-FractionFormat obj)
;;; 0
(vla-put-FractionFormat obj 1)
```

8.2.221. Свойство *Freeze*

Объект: **`Layer`**.

Описание: состояние заморозки слоя. Возможные значения:

- `:vlax-true` — заморожен;
- `:vlax-false` — разморожен.

Примечание

Текущий слой не может быть заморожен.

```
(setq obj (vla-Item (vla-get-Layers actdoc) "Layer1"))
;;; #<VLA-OBJECT IAcadLayer2 1827dc24>
(vla-get-Freeze obj)
;;; :vlax-false
(vla-put-Freeze obj :vlax-true)
```

8.2.222. Свойство *FullCRCValidation*

Объект: **`PreferencesOpenSave`**.

Описание: автоматический контроль объектов циклическим избыточным кодом (CRC — Cyclical Redundancy Check) при каждом обращении. Возможные значения:

- `:vlax-true` — включен;
- `:vlax-false` — выключен.

Примечание

Целесообразно применять только при выявлении проблем в чертеже, ввиду существенной ресурсоемкости.

```
(setq obj (vla-get-OpenSave
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesOpenSave 1829e7c4>
(vla-get-FullCRCValidation obj)
;;; :vlax-false
(vla-put-FullCRCValidation obj :vlax-true)
```

8.2.223. Свойство *FullFileName*

Объект: **FileDependency**.

Описание: полное имя внешнего файла. Только чтение. Возможные значения: строка (полное имя).

Примечание

При нахождении файла на путях поиска возвращается имя без пути.

```
(setq obj (vla-Item (vla-get-FileDependencies actdoc) 1))
;;; #<VLA-OBJECT IAcadFileDependency 1829d934>
(vla-get-FullFileName obj)
;;; "txt.shx"
;;; Файл на путях поиска.
(setq obj (vla-Item (vla-get-FileDependencies actdoc) 3))
;;; #<VLA-OBJECT IAcadFileDependency 1829dbe4>
(vla-get-FullFileName obj)
;;; "C:\\Documents and Settings\\ ... ats\\PC170048.tif"
;;; Файл не на путях поиска.
```

8.2.224. Свойство *FullName*

Объекты: **Application**, **Document**.

Описание: полное имя. Только чтение. Возможные значения: строка (полное имя).

```
(vla-get-FullName (vlax-get-acad-object))
;;; "C:\\Program Files\\Autodesk Building Systems 2005\\acad.exe"
(vla-get-FullName actdoc)
;;; "C:\\Temp\\Tutorial.dwg"
```

8.2.225. Свойство *FullScreenTrackingVector*

Объект: **PreferencesDrafting**.

Описание: длина векторов полярного слежения при включенном режиме полярного отслеживания. Значение сохраняется в системной переменной TRACKPATH. Возможные значения:

- `:vlax-true` — до границ экрана;
- `:vlax-false` — только до курсора.

```
(setq obj (vla-get-Drafting
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesDrafting 111cc838>
(vla-get-FullScreenTrackingVector obj)
;;; :vlax-true
(vla-put-FullScreenTrackingVector obj :vlax-false)
```

8.2.226. Свойство *GradientAngle*

Объект: **Hatch**.

Описание: угол заливки градиентной штриховки. Возможные значения: от 0 до 2π.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadHatch 11217744>
(vla-get-GradientAngle obj)
;;; 0.0
(vla-put-GradientAngle obj pi)
```

8.2.227. Свойство *GradientCentered*

Объект: **Hatch**.

Описание: центрирование градиентной штриховки. Возможные значения:

`:vlax-true` — центрированная;

`:vlax-false` — нецентрированная.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadHatch 11217744>
(vla-get-GradientCentered obj)
;;; :vlax-true
(vla-put-GradientCentered obj :vlax-false)
```

8.2.228. Свойство *GradientColor1*

Объект: **Hatch**.

Описание: первый цветовой компонент градиентной штриховки. Возможные значения: объект AcCmColor.

Примечание

Цвет должен иметь значение свойства ColorMethod, установленное как acColorMethodByACI (195) или acColorMethodByBlock (193). Подробнее см. свойство ColorMethod.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadHatch 11217744>
(vla-get-GradientColor1 obj)
;;; #<VLA-OBJECT IAcadAcCmColor 111c4f90>
(setq temp (vla-GetInterfaceObject (vlax-get-acad-object)
             "AutoCAD.AcCmColor.16"))
;;; #<VLA-OBJECT IAcadAcCmColor 11255850>
;;; Создаем новый цветовой объект.
(vla-put-ColorMethod temp acColorMethodByACI)
;;; Устанавливаем новому цвету свойство определения цвета.
(vla-put-EntityColor temp -1040122112)
;;; Назначаем код цвета.
(vla-put-GradientColor1 obj temp)
```

8.2.229. Свойство *GradientColor2*

Объект: **Hatch**.

Описание: второй цветовой компонент градиентной штриховки. Возможные значения: объект AcCmColor.

Примечание

Цвет должен иметь значение свойства ColorMethod, установленное как acColorMethodByACI (195) или acColorMethodByBlock (193). Подробнее см. свойство ColorMethod.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadHatch 11217744>
(vla-get-GradientColor2 obj)
;;; #<VLA-OBJECT IAcadAcCmColor 111c69a0>
(setq temp (vla-GetInterfaceObject (vlax-get-acad-object)
             "AutoCAD.AcCmColor.16"))
;;; #<VLA-OBJECT IAcadAcCmColor 11255850>
;;; Создаем новый цветовой объект.
```

```
(vla-put-ColorMethod temp acColorMethodByACI)
;;; Устанавливаем новому цвету свойство определения цвета.
(vla-put-EntityColor temp -1033768979)
;;; Назначаем код цвета.
(vla-put-GradientColor2 obj temp)
```

8.2.230. Свойство *GradientName*

Объект: **Hatch**.

Описание: тип (имя) градиентной заливки. Возможные значения:

- "LINEAR" — линейная;
- "CYLINDER" — цилиндрическая;
- "INVCYLINDER" — инверсно-цилиндрическая;
- "SPHERICAL" — сферическая;
- "HEMISPHERICAL" — полусферическая;
- "CURVED" — криволинейная;
- "INVSPHERICAL" — инверсно-сферическая;
- "INVHEMISPHERICAL" — инверсно-полусферическая;
- "INVCURVED" — инверсно-изогнутая.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadHatch 11217744>
(vla-get-GradientName obj)
;;; "SPHERICAL"
(vla-put-GradientName obj "CYLINDER")
```

8.2.231. Свойство *GraphicsWinLayoutBackgrndColor*

Объект: **PreferencesDisplay**.

Описание: цвет фона экрана в пространстве листа. Возможные значения: вариант типа OLE_COLOR.

Примечание

Этот тип данных не поддерживается AutoLISP, тем не менее управлять цветом можно, зная цифровые коды цветов, но нельзя получить значение установленного цвета фактически.

```
(setq obj (vla-get-Display
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesDisplay 0aebee8c>
```

```
(setq temp (vla-get-GraphicsWinLayoutBackgrndColor obj))
;;; #<variant 19 16777215>
(vlax-variant-value temp)
;;; error: LISPFromPtr failed. The type is not supported: 19
(vla-put-GraphicsWinLayoutBackgrndColor obj
  (vlax-make-variant 12634758 19))
```

8.2.232. Свойство *GraphicsWinModelBackgrndColor*

Объект: **PreferencesDisplay**.

Описание: цвет фона экрана в пространстве модели. Возможные значения: вариант типа OLE_COLOR.

Примечание

См. свойство GraphicsWinLayoutBackgrndColor.

```
(setq obj (vla-get-Display
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesDisplay 0aebee8c>
(setq temp (vla-get-GraphicsWinModelBackgrndColor obj))
;;; #<variant 19 0>
(vlax-variant-value temp)
;;; error: LISPFromPtr failed. The type is not supported: 19
(vla-put-GraphicsWinModelBackgrndColor obj
  (vlax-make-variant 12634758 19))
```

8.2.233. Свойство *Green*

Объект: **AcCmColor**.

Описание: содержание зеленой цветовой компоненты в цвете. Только чтение. Возможные значения: целое число от 0 до 255.

```
(setq obj (vla-GetInterfaceObject
            (vlax-get-acad-object) "AutoCAD.AcCmColor.16"))
;;; #<VLA-OBJECT IAcadAcCmColor 0aebe0b0>
(vla-get-Green obj)
;;; 0
```

8.2.234. Свойство *GridOn*

Объекты: **Viewport**, **PViewport**.

Описание: отображение сетки. Возможные значения:

- :vlax-true — включено;
- :vlax-false — выключено.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadPViewport2 0aed68d4>
(vla-get-GridOn obj)
;;; :vlax-false
(vla-put-GridOn obj :vlax-true)
```

8.2.235. Свойство *GripColorSelected*

Объект: **PreferencesSelection**.

Описание: цвет выбранных ручек. Соответствует системной переменной GRIPHOT. Возможные значения: целое число в диапазоне от 1 до 255, в том числе:

- `acRed` (1) — красный;
- `acYellow` (2) — желтый;
- `acGreen` (3) — зеленый;
- `acCyan` (4) — голубой;
- `acBlue` (5) — синий;
- `acMagenta` (6) — фиолетовый;
- `acWhite` (7) — белый.

```
(setq obj (vla-get-Selection
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesSelection 10bac5f0>
(vla-get-GripColorSelected obj)
;;; 1
(vla-put-GripColorSelected obj 24)
```

8.2.236. Свойство *GripColorUnselected*

Объект: **PreferencesSelection**.

Описание: цвет невыбранных ручек. Соответствует системной переменной GRIPCOLOR. Возможные значения: см. свойство `GripColorSelected`.

```
(setq obj (vla-get-Selection
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesSelection 10bac5f0>
(vla-get-GripColorUnselected obj)
;;; 160
(vla-put-GripColorUnselected obj 200)
```

8.2.237. Свойство *GripSize*

Объект: **PreferencesSelection**.

Описание: размер ручек в пикселах. Соответствует системной переменной GRIPSIZE. Возможные значения: целое число в диапазоне от 1 до 255.

```
(setq obj (vla-get-Selection
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesSelection 10bac5f0>
(vla-get-GripSize obj)
;;; 5
(vla-put-GripSize obj 20)
```

8.2.238. Свойство *Groups*

Объект: **Document**.

Описание: семейство групп. Возможные значения: VLA-объект семейства.

```
(vla-get-Groups actdoc)
;;; #<VLA-OBJECT IAcadGroups 10b9d744>
```

8.2.239. Свойство *Handle*

Объекты: **все графические примитивы**, **Block**, **Blocks**, **Dictionary**, **Dictionaries**, **DimStyle**, **DimStyles**, **Group**, **Groups**, **Layer**, **Layers**, **Layout**, **Layouts**, **Linetype**, **Linetypes**, **ModelSpace**, **PaperSpace**, **PlotConfiguration**, **PlotConfigurations**, **RegisteredApplication**, **RegisteredApplications**, **TextStyle**, **TextStyles**, **UCS**, **UCSs**, **View**, **Views**, **Viewport**, **Viewports**, **XRecord**.

Описание: метка объекта. Только чтение. Возможные значения: строка.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadEllipse 10ba6c44>
(vla-get-Handle obj)
;;; "417"
```

8.2.240. Свойство *HasAttributes*

Объекты: **BlockRef**, **ExternalReference**, **MInsertBlock**.

Описание: признак наличия у объекта атрибутов. Только чтение. Возможные значения:

- `:vlax-true` — есть;
- `:vlax-false` — нет.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadBlockReference 10ba56c4>
(vla-get-HasAttributes obj)
;;; :vlax-true
```

8.2.241. Свойство *HasExtensionDictionary*

Объекты: **все графические примитивы**, **AttributeReference**, **Dictionary**, **DimStyle**, **Group**, **Layer**, **Linetype**, **RegisteredApplication**, **TextStyle**, **UCS**, **View**, **Viewport**, **XRecord**.

Описание: наличие расширенного словаря. Только чтение. Возможные значения:

- `:vlax-true` — есть;
- `:vlax-false` — нет.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadBlockReference 10b98024>
(vla-get-HasExtensionDictionary obj)
;;; :vlax-false
```

8.2.242. Свойство *HasLeader*

Объект: **DimArcLength**.

Описание: признак наличия выноски, указывающей на образмериваемую дугу. Возможные значения:

- `:vlax-true` — есть;
- `:vlax-false` — нет.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimArcLength 08fe4b74>
(vla-get-HasLeader obj)
;;; :vlax-false
(vla-put-HasLeader obj :vlax-true)
```

8.2.243. Свойство *HasSheetView*

Объект: **PViewport**.

Описание: наличие вида, связанного с подшивкой. Только чтение. Возможные значения:

- `:vlax-true` — есть;
- `:vlax-false` — нет.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadPViewport2 1161d3f4>
(vla-get-HasSheetView obj)
;;; :vlax-false
```

8.2.244. Свойство *HasSubSelection*

2005

Объект: **Table**.

Описание: наличие выборки. Только чтение. Возможные значения:

- `:vlax-true` — есть;
- `:vlax-false` — нет.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 1161d324>
(vla-get-HasSubSelection obj)
;;; :vlax-false
```

8.2.245. Свойство *HasVpAssociation*

2005

Объект: **View**.

Описание: наличие ассоциативной связи с видовым экраном листа. Возможные значения:

- `:vlax-true` — есть;
- `:vlax-false` — нет.

```
(setq obj (vla-Item (vla-get-Views actdoc) 0))
;;; #<VLA-OBJECT IAcadView2 115da294>
(vla-get-HasVpAssociation obj)
;;; :vlax-false
(vla-put-HasVpAssociation obj :vlax-true)
```

8.2.246. Свойство *HatchObjectType*

Объект: **Hatch**.

Описание: тип штриховки (заливки) — обычная или градиентная. Возможные значения:

- `AcHatchObject` (0) — обычная (штриховка или заливка типа SOLID);
- `AcGradientObject` (1) — градиентная заливка.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadHatch 18a6abf4>
```

```
(vla-get-HatchObjectType obj)
;;; 0
(vla-put-HatchObjectType obj AcGradientObject)
```

8.2.247. Свойство *HatchStyle*

Объект: **Hatch**.

Описание: стиль обнаружения островков при штриховании. Возможные значения:

- `acHatchStyleNormal` (0) — нормальный;
- `acHatchStyleOuter` (1) — внешний;
- `acHatchStyleIgnore` (2) — игнорирующий.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadHatch 18a7ca84>
(vla-get-HatchStyle obj)
;;; 0
(vla-put-HatchStyle obj acHatchStyleIgnore)
```

8.2.248. Свойство *HeaderSuppressed*

Объекты: **Table**, **TableStyle**.

Описание: режим подавления заголовков (установка только для `TableStyle`). Возможные значения:

- `:vlax-true` — подавлять;
- `:vlax-false` — не подавлять.

```
(setq obj (vla-Item (vla-Item
     (vla-get-dictionaries actdoc) "acad_tablestyle") "Standard"))
;;; #<VLA-OBJECT IAcadTableStyle 18a95124>
(vla-get-HeaderSuppressed obj)
;;; :vlax-false
(vla-put-HeaderSuppressed obj :vlax-true)
```

8.2.249. Свойство *Height*

Объекты: **Application**, **Attribute**, **AttributeReference**, **MText**, **PViewport**, **Raster**, **Shape**, **Table**, **Text**, **TextStyle**, **Toolbar**, **Viewport**, **View**.

Описание: высота объекта (установка возможна для всех, кроме `Toolbar` и `Raster`). Возможные значения: неотрицательное число с плавающей точкой (кроме `Application` и `Toolbar`, для которых — целое).

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadText2 0ac93ae4>
(vla-get-Height obj)
;;; 20.0
(vla-put-Height obj 15)
```

8.2.250. Свойство *HelpFilePath*

Объект: **PreferencesFiles**.

Описание: полное имя файла справки. Возможные значения: строка с полным именем файла. Не проверяет наличие устанавливаемого файла.

```
(setq obj (vla-get-Files
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesFiles2 09b3a430>
(vla-get-HelpFilePath obj)
;;; "C:\\Program Files\\Autodesk B ... 5\\help\\abs2005.chm"
(vla-put-HelpFilePath obj "Y:\\Temp\\myfile.chm")
```

8.2.251. Свойство *HelpString*

Объекты: **PopupMenuItem**, **Toolbar**, **ToolbarItem**.

Описание: строка подсказки. Возможные значения: строка (установка пустой строки не допускается).

```
(setq obj (vla-Item (vla-Item (vla-get-Menus (vla-Item
  (vla-get-MenuGroups (vlax-get-acad-object)) "acad")) 9) 8))
;;; #<VLA-OBJECT IAcadPopupMenuItem 7ef2f0a8>
(vla-get-HelpString obj)
;;; "Duplicates objects:  COPY"
(vla-put-HelpString obj "Копирование объектов: COPY")
```

8.2.252. Свойство *HistoryLines*

Объект: **PreferencesDisplay**.

Описание: количество сохраняемых строк текстового окна. Возможные значения: целое число в диапазоне от 25 до 2048.

```
(setq obj (vla-get-Display
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesDisplay 09b3bcec>
(vla-get-HistoryLines obj)
;;; 400
(vla-put-HistoryLines obj 1000)
```

8.2.253. Свойство *HorizontalTextPosition*

Объекты: **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimDiametric**, **DimRadial**, **DimRadialLarge**, **DimRotated**.

Описание: тип горизонтального выравнивания и ориентации размерного текста. Переопределяет значение, установленное системной переменной DIMJUST. Возможные значения:

- `acHorzCentered` (0) — по центру;
- `acFirstExtensionLine` (1) — у первой выносной;
- `acSecondExtensionLine` (2) — у второй выносной;
- `acOverFirstExtension` (3) — вдоль первой выносной;
- `acOverSecondExtension` (4) — вдоль второй выносной.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRotated 09b52864>
(vla-get-HorizontalTextPosition obj)
;;; 0
(vla-put-HorizontalTextPosition obj acSecondExtensionLine)
```

8.2.254. Свойство *HorzCellMargin*

Объекты: **Table**, **TableStyle**.

Описание: горизонтальные поля ячеек. Возможные значения: неотрицательное вещественное число.

Примечание

При необходимости ячейки растягиваются, но не сжимаются.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTableStyle 18a95124>
(vla-get-HorzCellMargin obj)
;;; 1.5
(vla-put-HorzCellMargin obj 5)
```

8.2.255. Свойство *HWND*

Объекты: **Application**, **Document**.

Описание: дескриптор окна открытого документа. Только чтение. Возможные значения: целое число.

```
(vla-get-HWND (vlax-get-acad-object))
;;; 328222
```

8.2.256. Свойство *HyperlinkBase*

Объект: **SummaryInfo**.

Описание: база для относительных гиперссылок. Возможные значения: строка, в том числе пустая.

```
(setq obj (vla-get-SummaryInfo actdoc))
;;; #<VLA-OBJECT IAcadSummaryInfo 09b52714>
(vla-get-HyperlinkBase obj)
;;; ""
(vla-put-HyperlinkBase obj "//Server/Share/Folder")
```

8.2.257. Свойство *HyperlinkDisplayCursor*

Объект: **PreferencesUser**.

Описание: показ значка гиперссылки на курсоре. Возможные значения:

- `:vlax-true` — показывать;
- `:vlax-false` — не показывать.

```
(setq obj (vla-get-User
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesUser 09b7b4c8>
(vla-get-HyperlinkDisplayCursor obj)
;;; :vlax-true
(vla-put-HyperlinkDisplayCursor obj :vlax-false)
```

8.2.258. Свойство *HyperlinkDisplayTooltip*

Объект: **PreferencesUser**.

Описание: показ подсказки гиперссылки на курсоре. Возможные значения:

- `:vlax-true` — показывать;
- `:vlax-false` — не показывать.

Примечание

Установка не поддерживается.

```
(setq obj (vla-get-User
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesUser 09b7b4c8>
(vla-get-HyperlinkDisplayTooltip obj)
;;; :vlax-true
(vla-put-HyperlinkDisplayTooltip obj :vlax-false)
;;; error: Automation Error. Description was not provided.
```

8.2.259. Свойство *Hyperlinks*

Объекты: **все графические примитивы.**

Описание: семейство гиперссылок. Возможные значения: VLA-объект семейства.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadCircle 09b86f14>
(vla-get-Hyperlinks obj)
;;; #<VLA-OBJECT IAcadHyperlinks 09b80224>
```

8.2.260. Свойство *ImageFile*

Объект: **Raster.**

Описание: полное имя растрового файла. Возможные значения: строка с полным именем.

Примечание

Устанавливаемый файл должен быть поддерживаемого формата.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadRasterImage 09b8d924>
(vla-get-ImageFile obj)
;;; "C:\\Documents and ...\\PC170048.tif"
(vla-put-ImageFile obj "C:\\Temp\\new.tif")
```

8.2.261. Свойство *ImageFrameHighlight*

Объект: **PreferencesDisplay.**

Описание: тип подсветки растрового изображения при выборе. Соответствует системной переменной IMAGEHLT. Возможные значения:

- `:vlax-true` — подсвечивать весь растр;
- `:vlax-false` — подсвечивать только рамку.

```
(setq obj (vla-get-Display
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesDisplay 0adfa7ec>
(vla-get-ImageFrameHighlight obj)
;;; :vlax-false
(vla-put-ImageFrameHighlight obj :vlax-true)
```

8.2.262. Свойство *ImageHeight*

Объект: **Raster.**

Описание: высота растра в текущих единицах. Возможные значения: положительное вещественное число.

Примечание

В отличие от свойства `Height` оперирует текущими единицами и позволяет не только читать свойство, но устанавливать его. Пропорции растра при установке не сохраняются.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadRasterImage 09b8d924>
(vla-get-ImageHeight obj)
;;; 863.6
(vla-put-ImageHeight obj 1000)
```

8.2.263. Свойство *ImageVisibility*

Объект: **Raster**.

Описание: видимость растра. Возможные значения:

- `:vlax-true` — видимый;
- `:vlax-false` — невидимый (только рамка).

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadRasterImage 09b8d924>
(vla-get-ImageVisibility obj)
;;; :vlax-true
(vla-put-ImageVisibility obj :vlax-false)
```

8.2.264. Свойство *ImageWidth*

Объект: **Raster**.

Описание: ширина растра в текущих единицах. Возможные значения: положительное вещественное число.

Примечание

В отличие от свойства `Width` оперирует текущими единицами и позволяет не только читать свойство, но устанавливать его. Пропорции растра при установке не сохраняются.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadRasterImage 09b8d924>
(vla-get-ImageWidth obj)
;;; 1151.47
(vla-put-ImageWidth obj 1400)
```

8.2.265. Свойство *IncrementalSavePercent*

Объект: **PreferencesOpenSave**.

Описание: допустимое соотношение измененной части рисунка к сохраненной, при котором допускается сохранение только изменений. Соответствует системной переменной ISAVEPERCENT. Возможные значения: целое число в диапазоне от 0 до 100.

Примечание

При установке свойства в 0 каждое сохранение будет полным.

```
(setq obj (vla-get-OpenSave
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesOpenSave 09b954d4>
(vla-get-IncrementalSavePercent obj)
;;; 50
(vla-put-IncrementalSavePercent obj 75)
```

8.2.266. Свойство *Index*

Объекты: **FileDependency**, **PopupMenuItem**, **ToolbarItem**.

Описание: индекс (номер) объекта в семействе. Только чтение. Возможные значения: неотрицательное целое число.

```
(setq obj (vla-Item (vla-Item (vla-get-Menus (vla-Item
  (vla-get-MenuGroups (vlax-get-acad-object)) "acad")) 9) "ID_Copy"))
;;; #<VLA-OBJECT IAcadPopupMenuItem 7ef2f0a8>
(vla-get-Index obj)
;;; 8
```

8.2.267. Свойство *InsertionPoint*

Объекты: **Attribute**, **AttributeReference**, **BlockRef**, **ExternalReference**, **MInsertBlock**, **MText**, **Shape**, **Table**, **Text**, **Tolerance**.

Описание: точка вставки. Возможные значения: вариант (безопасный массив), содержащий список трех координат.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadText2 09b902f4>
(setq temp (vla-get-InsertionPoint obj))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (1808.2 -3.19122 0.0)
(vla-put-InsertionPoint obj (vlax-3d-point '(2000.0 -100.0 0.0)))
```

8.2.268. Свойство *InsUnits*

Объекты: **BlockRef**, **ExternalReference**, **MInsertBlock**.

Описание: единицы измерения блока. Только чтение. Возможные значения: строка.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadBlockReference2 02814e84>
(vla-get-InsUnits obj)
;;; "Millimeters"
```

8.2.269. Свойство *InsUnitsFactor*

Объекты: **BlockRef**, **ExternalReference**, **MInsertBlock**.

Описание: коэффициент пересчета единиц между описанием блока (файлом ссылки) и вхождением блока (внешней ссылки). Только чтение. Возможные значения: положительное вещественное число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadBlockReference2 02814e84>
(vla-get-InsUnitsFactor obj)
;;; 1.0
```

8.2.270. Свойство *Invisible*

Объекты: **Attribute**, **AttributeReference**.

Описание: невидимость атрибута. Переопределяет значение, установленное системной переменной AFLAGS. Возможные значения:

- `:vlax-true` — невидимый;
- `:vlax-false` — видимый.

```
(setq obj (vlax-ename->vla-object (car (nentsel))))
;;; #<VLA-OBJECT IAcadAttributeReference 09b903c4>
(vla-get-Invisible obj)
;;; :vlax-false
(vla-put-Invisible obj :vlax-true)
```

8.2.271. Свойство *IsCloned*

Объект: **IDPair**.

Описание: признак, был ли исходный объект клонирован в процессе копирования. Только чтение.

Возможные значения:

- `:vlax-true` — клонирован;
- `:vlax-false` — не клонирован.

Примечание

Специальный объект `IDPair` может быть получен только при использовании метода `CopyObjects`.

```
(setq actsel (vla-get-ActiveSelectionSet actdoc) tmp nil)
(vla-Clear actsel)
(vla-SelectOnScreen actsel)
(vla-CopyObjects actdoc (vlax-make-variant (vlax-safearray-fill
(vlax-make-safearray vlax-vbObject (cons 0 (1- (vla-get-Count actsel))))
  (vlax-for x actsel (setq tmp (cons x tmp)))))
    (vla-get-Block (vla-get-ActiveLayout actdoc)) 'idpairs)
(setq temp (vlax-safearray->list idpairs))
;;; (#<VLA-OBJECT IAcadIdPair 09b89274> #<VLA-OBJECT IAcadIdPair...)
(setq obj (car temp))
;;; #<VLA-OBJECT IAcadIdPair 09b89274>
(vla-get-IsCloned obj)
;;; :vlax-true
```

8.2.272. Свойство *IsDynamicBlock*

Объекты: **`Block`**, **`BlockRef`**, **`MInsertBlock`**.

Описание: признак динамического блока. Только чтение. Возможные значения:

- `:vlax-true` — да;
- `:vlax-false` — нет.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadBlockReference2 02814e84>
(vla-get-IsDynamicBlock obj)
;;; :vlax-true
```

8.2.273. Свойство *IsLayout*

Объект: **`Block`**.

Описание: определение, является ли объект на самом деле блоком или вкладкой пространства модели или листа (т. к. внутри рисунка пространство модели и вкладки пространства листа оформлены как блоки). Только чтение. Возможные значения:

- `:vlax-true` — является вкладкой;
- `:vlax-false` — не является вкладкой.

```
(setq obj (vla-Item (vla-get-Blocks actdoc) 3))
;;; #<VLA-OBJECT IAcadBlock2 09b9ebe4>
(vla-get-IsLayout obj)
;;; :vlax-false
```

8.2.274. Свойство *IsModified*

Объект: **FileDependency**.

Описание: признак, был ли внешний файл изменен. Только чтение. Возможные значения:

- `:vlax-true` — изменен;
- `:vlax-false` — не изменен.

```
(setq obj (vla-Item (vla-get-FileDependencies actdoc) 1))
;;; #<VLA-OBJECT IAcadFileDependency 09b9eae4>
(vla-get-IsModified obj)
;;; :vlax-false
```

8.2.275. Свойство *ISOPenWidth*

Объект: **Hatch**.

Описание: толщина линий штриховки по ISO в сотых долях миллиметра. Возможные значения: положительное число не более 2147483647.49999988079 или –1, в том числе:

- `acPenWidth013` (13) — 0.13 мм;
- `acPenWidth018` (18) — 0.18 мм;
- `acPenWidth025` (25) — 0.25 мм;
- `acPenWidth035` (35) — 0.35 мм;
- `acPenWidth070` (70) — 0.70 мм;
- `acPenWidth100` (100) — 1.00 мм;
- `acPenWidth140` (140) — 1.40 мм;
- `acPenWidth200` (200) — 2.00 мм;
- `acPenWidthUnk` (–1) — неизвестно.

Примечание

Если установлено значение не из списка, то при получении возвращается –1.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadHatch 09b9e934>
```

```
(vla-get-ISOPenWidth obj)
;;; 100
(vla-put-ISOPenWidth obj acPenWidth025)
```

8.2.276. Свойство *IsOwnerXlated*

Объект: **IDPair**.

Описание: определение того, был ли исходный объект транслирован в процессе копирования. Только чтение. Возможные значения:

- `:vlax-true` — транслирован;
- `:vlax-false` — не транслирован.

Примечание

Судя по документации, может принимать значение `:vlax-false` в случае, если владелец копируемого объекта уже существует. Актуально только при копировании между разными документами.

```
(setq actsel (vla-get-ActiveSelectionSet actdoc) tmp nil)
(vla-Clear actsel)
(vla-SelectOnScreen actsel)
(vla-CopyObjects actdoc (vlax-make-variant (vlax-safearray-fill
(vlax-make-safearray vlax-vbObject (cons 0 (1- (vla-get-Count actsel))))
(vlax-for x actsel (setq tmp (cons x tmp)))))
(vla-get-Block (vla-Item (vla-get-Layouts (vla-Item (vla-get-Documents
(vlax-get-acad-object)) "Drawing2.dwg")) "Model")) 'idpairs)
(setq temp (vlax-safearray->list idpairs))
;;; (#<VLA-OBJECT IAcadIdPair 09b50a34> #<VLA-OBJECT IAcadIdPair...)
(setq obj (car temp))
;;; #<VLA-OBJECT IAcadIdPair 09b50a34>
(vla-get-IsOwnerXlated obj)
;;; :vlax-true
```

8.2.277. Свойство *IsPartial*

Объект: **DimArcLength**.

Описание: признак частичного образмеривания дуги. Возможные значения:

- `:vlax-true` — да;
- `:vlax-false` — нет.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimArcLength 02814404>
(vla-get-IsPartial obj)
;;; :vlax-false
(vla-put-IsPartial obj :vlax-true)
```

8.2.278. Свойство *IsPeriodic*

Объект: **Spline**.

Описание: признак периодичности сплайна. Только чтение. Возможные значения:

- `:vlax-true` — периодический;
- `:vlax-false` — непериодический.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadSpline 09d51f24>
(vla-get-IsPeriodic obj)
;;; :vlax-false
```

8.2.279. Свойство *IsPlanar*

Объект: **Spline**.

Описание: признак, является ли сплайн плоским. Только чтение. Возможные значения:

- `:vlax-true` — плоский;
- `:vlax-false` — неплоский.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadSpline 09d51f24>
(vla-get-IsPlanar obj)
;;; :vlax-true
```

8.2.280. Свойство *IsPrimary*

Объект: **IDPair**.

Описание: определение того, был ли исходный объект членом первичного набора. Только чтение. Возможные значения:

- `:vlax-true` — да;
- `:vlax-false` — нет.

Примечание

Специальный объект `IDPair` может быть получен только при использовании метода `CopyObjects`.

```
(setq actsel (vla-get-ActiveSelectionSet actdoc) tmp nil)
(vla-Clear actsel)
(vla-SelectOnScreen actsel)
```

```
(vla-CopyObjects actdoc (vlax-make-variant (vlax-safearray-fill
(vlax-make-safearray vlax-vbObject (cons 0 (1- (vla-get-Count actsel))))
  (vlax-for x actsel (setq tmp (cons x tmp)))))
  (vla-get-Block (vla-get-ActiveLayout actdoc)) 'idpairs)
(setq temp (vlax-safearray->list idpairs))
;;; (#<VLA-OBJECT IAcadIdPair 0ad3ff84> #<VLA-OBJECT IAcadIdPair...)
(setq obj (car temp))
;;; #<VLA-OBJECT IAcadIdPair 0ad3ff84>
(vla-get-IsPrimary obj)
;;; :vlax-true
```

8.2.281. Свойство *IsQuiescent*

Объект: **AcadState**.

Описание: определение того, готов ли AutoCAD принять вызов внешнего процесса. Только чтение. Возможные значения:

- `:vlax-true` — да;
- `:vlax-false` — нет.

```
(setq obj (vla-GetAcadState (vlax-get-acad-object)))
;;; #<VLA-OBJECT IAcadState 0ad3fe90>
(vla-get-IsQuiescent obj)
;;; :vlax-false
```

8.2.282. Свойство *IsRational*

Объект: **Spline**.

Описание: признак, является ли сплайн рациональным. Только чтение. Возможные значения:

- `:vlax-true` — да;
- `:vlax-false` — нет.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadSpline 09d51f24>
(vla-get-IsRational obj)
;;; :vlax-false
```

8.2.283. Свойство *Issuer*

Объект: **SecurityParams**.

Описание: имя поставщика цифрового сертификата. Возможные значения: строка, в том числе пустая, значение не проверяется.

```
(setq obj (vla-GetInterfaceObject
            (vlax-get-acad-object) "AutoCAD.SecurityParams.16"))
;;; #<VLA-OBJECT IAcadSecurityParams 0ad3df64>
(vla-get-Issuer obj)
;;; ""
(vla-put-Issuer obj "Any string")
```

8.2.284. Свойство *IsXRef*

Объект: **Block**.

Описание: признак, является ли объект блоком или внешней ссылкой (т. к. блоки и внешние ссылки хранятся в одной таблице). Только чтение. Возможные значения:

- `:vlax-true` — внешняя ссылка;
- `:vlax-false` — не внешняя ссылка.

```
(setq obj (vla-Item (vla-get-Blocks actdoc) 3))
;;; #<VLA-OBJECT IAcadBlock2 0ad3e624>
(vla-get-IsXRef obj)
;;; :vlax-false
```

8.2.285. Свойство *JogAngle*

Объект: **DimRadialLarge**.

Описание: угол зигзага размерной линии. Возможные значения: вещественное число от $\pi/2$ до $\pi/32$.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRadialLarge 02820d94>
(vla-get-JogAngle obj)
;;; 1.5708
(vla-put-JogAngle obj (/ pi 4))
```

8.2.286. Свойство *JogLocation*

Объект: **DimRadialLarge**.

Описание: определяет местоположение зигзага на выноске. Возможные значения: координаты трехмерной точки в виде варианта с безопасным массивом.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRadialLarge 02820d94>
(setq temp (vla-get-JogLocation obj))
;;; #<variant 8197 ...>
```

```
(vlax-safearray->list (vlax-variant-value temp))
;;; (309.359 213.992 0.0)
(vla-put-JogLocation obj (vlax-3d-point (getpoint)))
```

8.2.287. Свойство *Justification*

Объект: **MLine**.

Описание: выравнивание относительно оси. Возможные значения:

- acTop (0) — верхнее;
- acZero (1) — по оси;
- acBottom (2) — нижнее.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadMLine2 027db2b4>
(vla-get-Justification obj)
;;; 0
(vla-put-Justification obj acBottom)
```

8.2.288. Свойство *Key*

Объект: **IDPair**.

Описание: ID исходного объекта. Только чтение. Возможные значения: положительное целое число.

Примечание

Специальный объект IDPair может быть получен только при использовании метода CopyObjects.

```
(setq actsel (vla-get-ActiveSelectionSet actdoc) tmp nil)
(vla-Clear actsel)
(vla-SelectOnScreen actsel)
(vla-CopyObjects actdoc (vlax-make-variant (vlax-safearray-fill
(vlax-make-safearray vlax-vbObject (cons 0 (1- (vla-get-Count actsel))))
  (vlax-for x actsel (setq tmp (cons x tmp)))))
  (vla-get-Block (vla-get-ActiveLayout actdoc)) 'idpairs)
(setq temp (vlax-safearray->list idpairs))
;;; (#<VLA-OBJECT IAcadIdPair 0ad44564> #<VLA-OBJECT IAcadIdPair...)
(setq obj (car temp))
;;; #<VLA-OBJECT IAcadIdPair 0ad44564>
(vla-get-Key obj)
;;; 2130137016
```

8.2.289. Свойство *KeyboardAccelerator*

Объект: **PreferencesUser**.

Описание: признак интерпретации комбинаций горячих клавиш в соответствии со стандартом Windows. Возможные значения:

- acPreferenceClassic (0) — классический (стандарт Windows);
- acPreferenceCustom (1) — пользовательский (стандарт старых версий AutoCAD).

```
(setq obj (vla-get-User
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesUser 0ad44538>
(vla-get-KeyboardAccelerator obj)
;;; 1
(vla-put-KeyboardAccelerator obj acPreferenceClassic)
```

8.2.290. Свойство *KeyboardPriority*

Объект: **PreferencesUser**.

Описание: приоритет ввода (влияние объектной привязки на координаты, введенные с клавиатуры). Соответствует системной переменной OSNAPCOORD. Возможные значения:

- acKeyboardRunningObjSnap (0) — объектная привязка;
- acKeyboardEntry (1) — клавиатурный ввод;
- acKeyboardEntryExceptScripts (2) — клавиатурный ввод, исключая пакеты (скрипты).

```
(setq obj (vla-get-User (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesUser 0ad44538>
(vla-get-KeyboardPriority obj)
;;; 2
(vla-put-KeyboardPriority obj acKeyboardRunningObjSnap)
```

8.2.291. Свойство *KeyDescriptions*

Объект: **LinkTemplate**.

Описание: ключевые поля. Только чтение. Возможные значения: VLA-объект.

```
(setq DBConn (vla-GetInterfaceObject (vlax-get-acad-object)
               "CAO.DbConnect.16"))
;;; #<VLA-OBJECT IDbConnect 04de13f0>
(setq DBLinksT (vlax-invoke DBConn 'GetLinkTemplates))
;;; #<VLA-OBJECT ILinkTemplates 04df12d0>
```

```
(setq DBLinkT (vla-Item DBLinksT "Link2"))
;;; #<VLA-OBJECT ILinkTemplate 04dff0d0>
(vlax-get-property DBLinkT 'KeyDescriptions)
;;; #<VLA-OBJECT IKeyDescriptions 0566a7a0>
```

8.2.292. Свойство *KeyLength*

Объект: **SecurityParams**.

Описание: длина ключа шифрования. Возможные значения: целое число, корректность определяется провайдером.

```
(setq obj (vla-GetInterfaceObject
            (vlax-get-acad-object) "AutoCAD.SecurityParams.16"))
;;; #<VLA-OBJECT IAcadSecurityParams 0ad45834>
(vla-get-KeyLength obj)
;;; 0
(vla-put-KeyLength obj 128)
```

8.2.293. Свойство *KeyValues*

Объект: **Link**.

Описание: значения ключей. Только чтение. Возможные значения: VLA-объект.

```
(setq DBConn (vla-GetInterfaceObject (vlax-get-acad-object)
               "CAO.DbConnect.16"))
;;; #<VLA-OBJECT IDbConnect 04de13f0>
(setq DBLinks (vlax-invoke DBConn 'GetLinks))
;;; #<VLA-OBJECT ILinks 04ece0c0>
(setq DBLink (vla-Item DBLinks 0))
;;; #<VLA-OBJECT ILink 04ecfd10>
(vlax-get-property DBLink 'KeyValues)
;;; #<VLA-OBJECT IKeyValues 04ecebe0>
```

8.2.294. Свойство *Keywords*

Объект: **SummaryInfo**.

Описание: список ключевых слов. Возможные значения: строка (может быть пустой).

```
(setq obj (vla-get-SummaryInfo actdoc))
;;; #<VLA-OBJECT IAcadSummaryInfo 09d51fd4>
(vla-get-Keywords obj)
;;; ""
(vla-put-Keywords obj "жилой дом проспект Космонавтов")
```

8.2.295. Свойство *Knots*

Объект: **Spline**.

Описание: координаты узлов направляющих векторов. Возможные значения: вариант (безопасный массив), содержащий двумерные координаты в виде плоского списка.

Примечание

Координаты точек в списке представлены в виде (X_1 Y_1 ... X_N Y_N), количество узлов на 1 больше, чем вершин.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadSpline 0ad57614>
(setq temp (vla-get-Knots obj))
;;; #<variant 8197 ...>
(setq temp (vlax-safearray->list (vlax-variant-value temp)))
;;; (0.0 0.0 0.0 0.0 448.021 791.127 1085.64 1085.64 1085.64 1085.64)
(setq temp (subst 500.0 (nth 5 temp) temp))
;;; Заменяем значение Y третьего узла.
;;; (0.0 0.0 0.0 0.0 448.021 500.0 1085.64 1085.64 1085.64 1085.64)
(vla-put-Knots obj (vlax-safearray-fill
    (vlax-make-safearray vlax-vbDouble '(0 . 9)) temp))
```

8.2.296. Свойство *Label*

Объекты: **Link**, **PopupMenuItem**.

Описание: метка или ярлык. Возможные значения: для `PopupMenuItem` — строка, для **Link** — VLA-объект.

```
(setq obj (vla-Item (vla-Item (vla-get-Menus (vla-Item
(vla-get-MenuGroups (vlax-get-acad-object)) "acad")) 9) 8))
;;; #<VLA-OBJECT IAcadPopupMenuItem 7ef2f0a8>
(vla-get-Label obj)
;;; "Cop&y"
(vla-put-Label obj "Копировать") ;_ для PopupMenuItem
(setq DBConn (vla-GetInterfaceObject (vlax-get-acad-object)
               "CAO.DbConnect.16"))
;;; #<VLA-OBJECT IDbConnect 04de13f0>
(setq DBLinksT (vlax-invoke DBConn 'GetLinkTemplates))
;;; #<VLA-OBJECT ILinkTemplates 04df12d0>
(setq DBLinkT (vla-Item DBLinksT "Link2"))
;;; #<VLA-OBJECT ILinkTemplate 04dff0d0>
(setq Links (vlax-invoke-method DBConn 'GetLinks DBLinkT
  (vlax-make-variant (vlax-safearray-fill (vlax-make-safearray
```

```
    vlax-vbLong '(0 . 0)) (list (vla-get-ObjectID (vlax-ename->vla-object
      (car (entsel))))))) 2 actdoc))
;;; #<VLA-OBJECT ILinks 0566baf0>
(setq Link (vla-Item Links 1))
;;; #<VLA-OBJECT ILink 0566fd30>
(setq obj (vla-get-Label Link))
;;; #<VLA-OBJECT ILabel 0566d620> ;_ для Link
```

8.2.297. Свойство *LabelBlockId*

2005

Объект: **PViewport**.

Описание: ID привязанного блока с маркой вида (в подшивках). Возможные значения: целое число.

Примечание

Свойство может быть недоступно.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadPViewport2 2228a1c4>
(vla-get-LabelBlockId obj)
;;; 1848946576
(vla-ObjectIDToObject actdoc (vla-get-LabelBlockId obj))
;;; #<VLA-OBJECT IAcadBlockReference 22297ca4>
(setq temp (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadBlockReference 2228e724>
(vla-put-LabelBlockId obj (vla-get-ObjectID temp))
```

8.2.298. Свойство *LabelTemplate*

Объект: **Label**.

Описание: шаблон ярлыка. Только чтение. Возможные значения: VLA-объект.

```
(setq DBConn (vla-GetInterfaceObject (vlax-get-acad-object)
               "CAO.DbConnect.16"))
;;; #<VLA-OBJECT IDbConnect 04de13f0>
(setq DBLinksT (vlax-invoke DBConn 'GetLinkTemplates))
;;; #<VLA-OBJECT ILinkTemplates 04ede6a0>
(setq DBLinkT (vla-Item DBLinksT "Link2"))
;;; #<VLA-OBJECT ILinkTemplate 04ede4d0>
(setq Links (vlax-invoke-method DBConn 'GetLinks DBLinkT
  (vlax-make-variant (vlax-safearray-fill (vlax-make-safearray
    vlax-vbLong '(0 . 0)) (list (vla-get-ObjectID (vlax-ename->vla-object
      (car (entsel))))))) 2 actdoc))
;;; #<VLA-OBJECT ILinks 04edf630>
```

```
(setq Link (vla-Item Links 1))
;;; #<VLA-OBJECT ILink 04edadf0>
(setq obj (vla-get-Label Link))
;;; #<VLA-OBJECT ILabel 04edadd0>
(vlax-get-property obj 'LabelTemplate) ;_ получение, как свойства
;;; #<VLA-OBJECT ILabelTemplate 04edaad0>
(setq DBLablsT (vlax-invoke DBConn 'GetLabelTemplates))
;;; #<VLA-OBJECT ILabelTemplates 04edf9d0>
(vla-Item DBLablsT 0) ;_ получение, как объекта из коллекции
;;; #<VLA-OBJECT ILabelTemplate 04ede070>
```

8.2.299. Свойство *LargeButtons*

Объекты: **Toolbar**, **Toolbars**.

Описание: режим больших кнопок (установка только для `Toolbars`). Возможные значения:

- `:vlax-true` — включен;
- `:vlax-false` — выключен.

```
(setq obj (vla-get-Toolbars (vla-Item (vla-get-MenuGroups
    (vlax-get-acad-object)) "acad")))
;;; #<VLA-OBJECT IAcadToolbars 10eb48f4>
(vla-get-LargeButtons obj)
;;; :vlax-false
(vla-put-LargeButtons obj :vlax-true)
```

8.2.300. Свойство *LastHeight*

Объект: **TextStyle**.

Описание: высота текста, примененная последней. Возможные значения: положительное вещественное число.

```
(setq obj (vla-Item (vla-get-TextStyles actdoc) "Standard"))
;;; #<VLA-OBJECT IAcadTextStyle 10eb6cb4>
(vla-get-LastHeight obj)
;;; 20.0
(vla-put-LastHeight obj 3)
```

8.2.301. Свойство *LastSavedBy*

Объект: **SummaryInfo**.

Описание: имя автора документа. Возможные значения: строка с именем или пустая строка.

```
(setq obj (vla-get-SummaryInfo actdoc))
;;; #<VLA-OBJECT IAcadSummaryInfo 10eb6fa4>
(vla-get-LastSavedBy obj)
;;; "Alaspher"
(vla-put-LastSavedBy obj "Пупкин В.В.")
```

8.2.302. Свойство *Layer*

Объекты: **все графические объекты**, **AttributeReference**, **Group**.

Описание: имя слоя (для установки слой должен существовать). Возможные значения: строка с именем слоя.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadCircle 10ebd9f4>
(vla-get-Layer obj)
;;; "Layer2"
(vla-put-Layer obj "Layer1")
```

8.2.303. Свойство *LayerOn*

Объект: **Layer**.

Описание: признак того, что слой включен. Возможные значения:

- `:vlax-true` — включен;
- `:vlax-false` — выключен.

```
(setq obj (vla-Item (vla-get-Layers actdoc) "Layer1"))
;;; #<VLA-OBJECT IAcadLayer2 10cd8ab4>
(vla-get-LayerOn obj)
;;; :vlax-true
(vla-put-LayerOn obj :vlax-false)
```

8.2.304. Свойство *Layers*

Объект: **Document**.

Описание: указатель на семейство слоев документа. Только чтение. Возможные значения: VLA-объект (указатель на семейство).

```
(vla-get-Layers actdoc)
;;; #<VLA-OBJECT IAcadLayers2 10cd8cf4>
```

8.2.305. Свойство *LayerState*

2005

Объект: **View**.

Описание: имя конфигурации слоев. Возможные значения: строка (может быть пустой).

```
(setq obj (vla-Item (vla-get-Views actdoc) 0))
;;; #<VLA-OBJECT IAcadView2 10cda7a4>
(vla-get-LayerState obj)
;;; "ACAD_VIEWS_test"
(vla-put-LayerState obj "ACAD_VIEWS_v1")
```

8.2.306. Свойство *Layout*

Объекты: **ModelSpace**, **PaperSpace**, **Block**.

Описание: связанная вкладка. Только чтение. Возможные значения: VLA-объект.

```
(setq obj (vla-get-ModelSpace actdoc))
;;; #<VLA-OBJECT IAcadModelSpace2 10ce2074>
(vla-get-Layout obj)
;;; #<VLA-OBJECT IAcadLayout 10cd59f4>
```

8.2.307. Свойство *LayoutCreateViewport*

Объект: **PreferencesDisplay**.

Описание: автоматическое создание плавающего видового экрана при создании нового листа. Возможные значения:

- `:vlax-true` — включено;
- `:vlax-false` — выключено.

```
(setq obj (vla-get-Display
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesDisplay 10cd59ac>
(vla-get-LayoutCreateViewport obj)
;;; :vlax-true
(vla-put-LayoutCreateViewport obj :vlax-false)
```

8.2.308. Свойство *LayoutCrosshairColor*

Объект: **PreferencesDisplay**.

Описание: цвет курсора в пространстве листа. Возможные значения: вариант типа `OLE_COLOR`.

Примечание

Этот тип данных не поддерживается в AutoLISP. Тем не менее управлять цветом можно, зная цифровые коды цветов, но нельзя получить значение фактически установленного цвета.

```
(setq obj (vla-get-Display
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesDisplay 10cd119c>
```

```
(setq temp (vla-get-LayoutCrosshairColor obj))
;;; #<variant 19 0>
(vlax-variant-value temp)
;;; error: LISPFromPtr failed. The type is not supported: 19
(vla-put-LayoutCrosshairColor obj (vlax-make-variant 12634758 19))
```

8.2.309. Свойство *LayoutDisplayMargins*

Объект: **PreferencesDisplay**.

Описание: отображение полей печатного листа. Возможные значения:

- `:vlax-true` — включено;
- `:vlax-false` — выключено.

```
(setq obj (vla-get-Display
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesDisplay 10cd59ac>
(vla-get-LayoutDisplayMargins obj)
;;; :vlax-true
(vla-put-LayoutDisplayMargins obj :vlax-false)
(vla-Regen actdoc acActiveViewport)
```

8.2.310. Свойство *LayoutDisplayPaper*

Объект: **PreferencesDisplay**.

Описание: отображение границ печатного листа. Возможные значения:

- `:vlax-true` — включено;
- `:vlax-false` — выключено.

```
(setq obj (vla-get-Display
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesDisplay 10cd59ac>
(vla-get-LayoutDisplayPaper obj)
;;; :vlax-true
(vla-put-LayoutDisplayPaper obj :vlax-false)
(vla-Regen actdoc acActiveViewport)
```

8.2.311. Свойство *LayoutDisplayPaperShadow*

Объект: **PreferencesDisplay**.

Описание: отображение тени от печатного листа. Возможные значения:

- `:vlax-true` — включено;
- `:vlax-false` — выключено.

```
(setq obj (vla-get-Display
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesDisplay 10cd59ac>
(vla-get-LayoutDisplayPaperShadow obj)
;;; :vlax-true
(vla-put-LayoutDisplayPaperShadow obj :vlax-false)
(vla-Regen actdoc acActiveViewport)
```

8.2.312. Свойство *LayoutId*

2005

Объект: **View**.

Описание: ID связанного листа. Только чтение. Возможные значения: положительное целое число или 0, если нет.

```
(setq obj (vla-Item (vla-get-Views actdoc) 0))
;;; #<VLA-OBJECT IAcadView2 10cd22c4>
(vla-get-LayoutId obj)
;;; 0
```

8.2.313. Свойство *Layouts*

Объект: **Document**.

Описание: указатель на семейство вкладок документа. Только чтение. Возможные значения: VLA-объект (указатель на семейство).

```
(vla-get-Layouts actdoc)
;;; #<VLA-OBJECT IAcadLayouts 10cd2224>
```

8.2.314. Свойство *LayoutShowPlotSetup*

Объект: **PreferencesDisplay**.

Описание: автоматический показ диалога настройки печати при создании новой вкладки листа. Возможные значения:

- `:vlax-true` — включен;
- `:vlax-false` — выключен.

```
(setq obj (vla-get-Display
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesDisplay 10ce250c>
(vla-get-LayoutShowPlotSetup obj)
;;; :vlax-false
(vla-put-LayoutShowPlotSetup obj :vlax-true)
```

8.2.315. Свойство *Leader1Point*

2006

Объект: **DimArcLength**.

Описание: точка первой выноски. Возможные значения: координаты трехмерной точки в виде варианта с безопасным массивом.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimArcLength 027f4ed4>
(setq temp (vla-get-Leader1Point obj))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (338.284 163.261 0.0)
(vla-put-Leader1Point obj (vlax-3d-point (getpoint)))
```

8.2.316. Свойство *Leader2Point*

2006

Объект: **DimArcLength**.

Описание: точка второй выноски. Возможные значения: координаты трехмерной точки в виде варианта с безопасным массивом.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimArcLength 027f4ed4>
(setq temp (vla-get-Leader2Point obj))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (333.463 154.354 0.0)
(vla-put-Leader2Point obj (vlax-3d-point (getpoint)))
```

8.2.317. Свойство *LeaderAttachment*

Объект: **LabelTemplate**.

Описание: выравнивание точки привязки выноски. Возможные значения: см. свойство `AttachmentPoint`.

```
(setq DBConn (vla-GetInterfaceObject (vlax-get-acad-object)
              "CAO.DbConnect.16"))
;;; #<VLA-OBJECT IDbConnect 0573c770>
(setq DBLablsT (vlax-invoke DBConn 'GetLabelTemplates))
;;; #<VLA-OBJECT ILabelTemplates 0573cc10>
(setq DBLablT (vla-Item DBLablsT 0)) ;_был создан заранее
;;; #<VLA-OBJECT ILabelTemplate 0573d010>
(vlax-get-property DBLablT 'LeaderAttachment)
;;; 5
(vlax-put-property DBLablT 'LeaderAttachment acAttachmentPointTopLeft)
```

8.2.318. Свойство *LeaderLength*

Объекты: **DimDiametric**, **DimRadial**.

Описание: расстояние от стрелки до начала текста. Возможные значения: вещественное число (отрицательные — внутрь, положительные — наружу).

Примечание

Этот тип данных не поддерживает чтение. Кроме того, устанавливать его можно только в случае, если текст размещался автоматически, и только один раз, на последующие установки объект не реагирует.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimDiametric 10eca314>
(vla-put-LeaderLength obj 40.0)
```

8.2.319. Свойство *LeaderObjectId*

Объект: **Label**.

Описание: ID объекта, к которому относится ярлык. Только чтение. Возможные значения: положительное целое.

```
(setq DBConn (vla-GetInterfaceObject (vlax-get-acad-object)
              "CAO.DbConnect.16"))
;;; #<VLA-OBJECT IDbConnect 0573c770>
(setq DBLinksT (vlax-invoke DBConn 'GetLinkTemplates))
;;; #<VLA-OBJECT ILinkTemplates 0573bbf0>
(setq DBLinkT (vla-Item DBLinksT "Link2"))
;;; #<VLA-OBJECT ILinkTemplate 0573d200>
(setq Links (vlax-invoke-method DBConn 'GetLinks DBLinkT
  (vlax-make-variant (vlax-safearray-fill (vlax-make-safearray
    vlax-vbLong '(0 . 0)) (list (vla-get-ObjectID (vlax-ename->vla-object
      (car (entsel))))))) 2 actdoc))
;;; #<VLA-OBJECT ILinks 05739be0>
(setq Link (vla-Item Links 1))
;;; #<VLA-OBJECT ILink 0573eaa0>
(setq obj (vla-get-Label Link))
;;; #<VLA-OBJECT ILabel 0573e050>
(vlax-get-property obj 'LeaderObjectId)
;;; 2130124472
```

8.2.320. Свойство *LeaderOffset*

Объект: **LabelTemplate**.

Описание: смещение точки привязки выноски. Возможные значения: координаты трехмерной точки в виде варианта, содержащего безопасный массив.

```
(setq DBConn (vla-GetInterfaceObject (vlax-get-acad-object)
              "CAO.DbConnect.16"))
;;; #<VLA-OBJECT IDbConnect 0573c770>
(setq DBLablsT (vlax-invoke DBConn 'GetLabelTemplates))
;;; #<VLA-OBJECT ILabelTemplates 0573cc10>
(setq DBLablT (vla-Item DBLablsT 0)) ;_был создан заранее
;;; #<VLA-OBJECT ILabelTemplate 0573d010>
(setq tmp (vlax-get-property DBLablT 'LeaderOffset))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value tmp))
;;; (1.0 1.0 0.0)
(vlax-put-property DBLablT 'LeaderOffset (vlax-3d-point '(1.2 2.0 0.0)))
```

8.2.321. Свойство *Left*

Объект: **Toolbar**.

Описание: расстояние от левого края экрана до левого края плавающей панели инструментов в пикселах. Возможные значения: неотрицательное целое число.

```
(setq obj (vla-Item (vla-get-Toolbars (vla-Item (vla-get-MenuGroups
        (vlax-get-acad-object)) "acad")) "Layers"))
;;; #<VLA-OBJECT IAcadToolbar 0101275c>
(vla-get-Left obj)
;;; 217
(vla-put-Left obj 400)
```

8.2.322. Свойство *Length*

Объекты: **Line**, **LightweightPolyline**, **Polyline**, **3dPolyline**.

Описание: длина. Только чтение. Возможные значения: неотрицательное вещественное число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadLWPolyline 103d1014>
(vla-get-Length obj)
;;; 863.367
```

8.2.323. Свойство *LensLength*

Объект: **PViewport**.

Описание: фокусное расстояние в перспективном виде. Возможные значения: положительное вещественное число.

Примечание

Формально данному свойству может быть задано нулевое или даже отрицательное значение, что, однако, лишено физического смысла.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadPViewport2 103d06d4>
(vla-get-LensLength obj)
;;; 50.0
(vla-put-LensLength obj 28)
```

8.2.324. Свойство *Limits*

Объект: **Document**.

Описание: лимиты. Соответствует системным переменным LIMMIN и LIMMAX.

Возможные значения: вариант с типом безопасного массива, содержащий двумерные координаты в виде плоского списка. При установке допускается задавать безопасный массив.

Примечание

Координаты в списке представлены в виде: (X_{min} Y_{min} X_{max} Y_{max}).

```
(setq temp (vla-get-Limits actdoc))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (0.0 0.0 420.0 297.0)
(vla-put-Limits actdoc (vlax-safearray-fill
    (vlax-make-safearray vlax-vbDouble '(0 . 3)) '(-150 -100 840 600)))
```

8.2.325. Свойство *LinearScaleFactor*

Объекты: **DimAligned**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**.

Описание: линейный масштабный коэффициент. Переопределяет значение, установленное системной переменной DIMLFAC. Возможные значения: вещественное число, кроме 0.

Примечание

Отрицательная величина используется для указания того, что коэффициент действует только в пространстве листа.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimDiametric 1886d314>
(vla-get-LinearScaleFactor obj)
;;; -4.03127
(vla-put-LinearScaleFactor obj 3)
```

8.2.326. Свойство *LineSpacingDistance*

Объект: **MText**.

Описание: межстрочный интервал в единицах чертежа. Возможные значения: вещественное число.

Примечание

При установке данного свойства следует учитывать, что его величина не может быть такой, чтобы свойство `LineSpacingFactor` вышло из допустимого диапазона.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadMText2 18868c64>
(vla-get-LineSpacingDistance obj)
;;; 4.16667
(vla-put-LineSpacingDistance obj 8)
```

8.2.327. Свойство *LineSpacingFactor*

Объект: **MText**.

Описание: относительный межстрочный интервал. Возможные значения: вещественное число в диапазоне от 0.25 до 4.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadMText2 18868c64>
(vla-get-LineSpacingFactor obj)
;;; 1.6
(vla-put-LineSpacingFactor obj 1)
```

8.2.328. Свойство *LineSpacingStyle*

Объект: **MText**.

Описание: стиль задания межстрочного интервала. Возможные значения:

- `acLineSpacingStyleAtLeast` (1) — минимум;
- `acLineSpacingStyleExactly` (2) — точно.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadMText2 18868c64>
(vla-get-LineSpacingStyle obj)
;;; 1
(vla-put-LineSpacingStyle obj acLineSpacingStyleExactly)
```

8.2.329. Свойство *Linetype*

Объекты: **все графические объекты**, **AttributeReference**, **Group**, **Layer**, **LabelTemplate**.

Описание: тип линий (получение невозможно для `Group`). Возможные значения: строка с именем типа линий (тип должен быть загружен).

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadCircle 18898894>
(vla-get-Linetype obj)
;;; "ByLayer"
(vla-put-Linetype obj "center")
```

8.2.330. Свойство *LinetypeGeneration*

Объекты: **LightweightPolyline**, **Polyline**.

Описание: непрерывность генерации несплошных типов линий на узловых точках. Возможные значения:

- `:vlax-true` — включена;
- `:vlax-false` — выключена.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadLWPolyline 188a37c4>
(vla-get-LinetypeGeneration obj)
;;; :vlax-false
(vla-put-LinetypeGeneration obj :vlax-true)
```

8.2.331. Свойство *Linetypes*

Объект: **Document**.

Описание: семейство типов линий. Только чтение. Возможные значения: указатель на VLA-объект семейства.

```
(vla-get-Linetypes actdoc)
;;; #<VLA-OBJECT IAcadLineTypes 188a6244>
```

8.2.332. Свойство *LinetypeScale*

Объекты: **все графические объекты**, **AttributeReference**, **Group**.

Описание: линейный масштабный коэффициент (получение невозможно для `Group`). Возможные значения: неотрицательное вещественное число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadLWPolyline 188a37c4>
```

```
(vla-get-LinetypeScale obj)
;;; 1.0
(vla-put-LinetypeScale obj 2)
```

8.2.333. Свойство *LineWeight*

Объекты: **все графические объекты**, **DatabasePreferences**, **Layer**.

Описание: вес линий. Значение по умолчанию хранится в системной переменной LWDEFAULT. Возможные значения: см. свойство `DimensionLineWeight`.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadLWPolyline 188a37c4>
(vla-get-LineWeight obj)
;;; -1
(vla-put-LineWeight obj acLnWt025)
```

8.2.334. Свойство *LineWeightDisplay*

Объект: **DatabasePreferences**.

Описание: отображение весов линий на экране. Возможные значения:

- `:vlax-true` — включено;
- `:vlax-false` — выключено.

```
(setq obj (vla-get-Preferences actdoc))
(vla-get-LineWeightDisplay obj)
;;; :vlax-false
(vla-put-LineWeightDisplay obj :vlax-true)
```

8.2.335. Свойство *Link*

Объект: **Label**.

Описание: ссылка, к которой относится ярлык. Только чтение. Возможные значения: VLA-объект.

```
(setq DBConn (vla-GetInterfaceObject (vlax-get-acad-object)
              "CAO.DbConnect.16"))
;;; #<VLA-OBJECT IDbConnect 0573c770>
(setq DBLinksT (vlax-invoke DBConn 'GetLinkTemplates))
;;; #<VLA-OBJECT ILinkTemplates 0573bbf0>
(setq DBLinkT (vla-Item DBLinksT "Link2"))
;;; #<VLA-OBJECT ILinkTemplate 0573d200>
(setq Links (vlax-invoke-method DBConn 'GetLinks DBLinkT
  (vlax-make-variant (vlax-safearray-fill (vlax-make-safearray
```

```
    vlax-vbLong '(0 . 0)) (list (vla-get-ObjectID (vlax-ename->vla-object
      (car (entsel))))))) 2 actdoc))
;;; #<VLA-OBJECT ILinks 05739be0>
(setq Link (vla-Item Links 1))
;;; #<VLA-OBJECT ILink 0573eaa0>
(setq obj (vla-get-Label Link))
;;; #<VLA-OBJECT ILabel 0573e050>
(vlax-get-property obj 'Link)
;;; #<VLA-OBJECT ILink 05740210>
```

8.2.336. Свойство *LinkTemplate*

Объект: **Link**.

Описание: шаблон ссылки. Только чтение. Возможные значения: VLA-объект.

```
(setq DBConn (vla-GetInterfaceObject (vlax-get-acad-object)
              "CAO.DbConnect.16"))
;;; #<VLA-OBJECT IDbConnect 0573c770>
(setq DBLinks (vlax-invoke DBConn 'GetLinks))
;;; #<VLA-OBJECT ILinks 05742d90>
(setq DBLink (vla-Item DBLinks 0))
;;; #<VLA-OBJECT ILink 05744ee0>
(vlax-get-property DBLink 'LinkTemplate)
;;; #<VLA-OBJECT ILinkTemplate 05740260>
```

8.2.337. Свойство *LinkType*

Объект: **Link**.

Описание: тип ссылки. Только чтение. Возможные значения:

- `kUnkownLinkType` (1) — неизвестный;
- `kEntityLinkType` (2) — связанный, без ярлыка;
- `kFSLabelType` (4) — свободный ярлык;
- `kAttachedLabelType` (8) — связанный ярлык.

Примечание

В текущих версиях Visual LISP данные символы констант не поддерживаются.

```
(setq DBConn (vla-GetInterfaceObject (vlax-get-acad-object)
              "CAO.DbConnect.16"))
;;; #<VLA-OBJECT IDbConnect 0573c770>
(setq DBLinks (vlax-invoke DBConn 'GetLinks))
;;; #<VLA-OBJECT ILinks 05742d90>
```

```
(setq DBLink (vla-Item DBLinks 0))
;;; #<VLA-OBJECT ILink 05744ee0>
(vlax-get-property DBLink 'LinkType)
;;; 2
```

8.2.338. Свойство *LoadAcadLspInAllDocuments*

Объект: **PreferencesSystem**.

Описание: признак загрузки файла acad.lsp с каждым DWG-файлом. Хранится в системной переменной ACADLSPASDOC. Возможные значения:

- `:vlax-true` — да;
- `:vlax-false` — нет.

```
(setq obj (vla-get-System
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesSystem 0516a294>
(vla-get-LoadAcadLspInAllDocuments obj)
;;; :vlax-false
(vla-put-LoadAcadLspInAllDocuments obj :vlax-true)
```

8.2.339. Свойство *LocaleId*

Объект: **Application**.

Описание: локальный ID, формируемый операционной системой, для текущей сессии. Только чтение. Возможные значения: положительное целое число.

```
(setq obj (vlax-get-acad-object))
;;; #<VLA-OBJECT IAcadApplication 00b9d604>
(vla-get-LocaleId obj)
;;; 1033
```

8.2.340. Свойство *Lock*

Объект: **Layer**.

Описание: блокировка. Возможные значения:

- `:vlax-true` — слой блокирован;
- `:vlax-false` — слой разблокирован.

```
(setq obj (vla-Item (vla-get-Layers actdoc) "Layer1"))
;;; #<VLA-OBJECT IAcadLayer2 134060f4>
(vla-get-Lock obj)
;;; :vlax-false
(vla-put-Lock obj :vlax-true)
```

8.2.341. Свойство *LockAspectRatio*

2005

Объект: **Ole**.

Описание: блокировка пропорций. Возможные значения:

- `:vlax-true` — пропорции блокированы;
- `:vlax-false` — пропорции не блокированы.

Примечание

Свойство не документировано. Не исключено, что присутствует и у других объектов.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadOle 0aecf254>
(vla-get-LockAspectRatio obj)
;;; :vlax-true
(vla-put-LockAspectRatio obj :vlax-false)
```

8.2.342. Свойство *LockPosition*

2006

Объекты: **Attribute**, **AttributeReference**.

Описание: блокировка перемещения атрибута относительно блока. Для `Attribute` — только чтение. Возможные значения:

- `:vlax-true` — включена;
- `:vlax-false` — выключена.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadAttribute2 02811534>
(vla-get-LockPosition obj)
;;; :vlax-false
(vla-put-LockPosition obj :vlax-true)
```

8.2.343. Свойство *LogFileOn*

Объект: **PreferencesOpenSave**.

Описание: запись в LOG-файл всего вывода, направленного в текстовое окно. Соответствует системной переменной LOGFILEMODE. Возможные значения:

- `:vlax-true` — включена;
- `:vlax-false` — выключена.

```
(setq obj (vla-get-OpenSave
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesOpenSave 0aef8e94>
```

```
(vla-get-LogFileOn obj)
;;; :vlax-false
(vla-put-LogFileOn obj :vlax-true)
```

8.2.344. Свойство *LogFilePath*

Объект: **PreferencesFiles**.

Описание: путь сохранения протокола (LOG-файла). Соответствует системной переменной LOGFILEPATH. Возможные значения: строка с полным путем.

Примечание

При установке путь должен существовать.

```
(setq obj (vla-get-Files
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesFiles2 0aef8e40>
(vla-get-LogFilePath obj)
;;; "C:\\Documents and ... ABS 2005\\enu\\"
(vla-put-LogFilePath obj "C:\\Temp\\")
```

8.2.345. Свойство *LowerLeftCorner*

Объект: **Viewport**.

Описание: координаты левого нижнего угла в долях экрана. Только чтение. Возможные значения: вариант (безопасный массив), содержащий двумерные координаты.

```
(setq obj (vla-Item (vla-get-Viewports actdoc) 0))
;;; #<VLA-OBJECT IAcadViewport 0af00014>
(setq temp (vla-get-LowerLeftCorner obj))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (0.0 0.0)
```

8.2.346. Свойство *Macro*

Объекты: **PopupMenuItem**, **ToolbarItem**.

Описание: строка макроса, связанного с элементом меню. Возможные значения: непустая строка.

Примечание

После экспериментов не забудьте восстановить исходное значение.

```
(setq obj (vla-Item (vla-Item (vla-get-Menus (vla-Item
  (vla-get-MenuGroups (vlax-get-acad-object)) "acad")) 9) 10))
;;; #<VLA-OBJECT IAcadPopupMenuItem 7ef2f128>
(vla-get-Macro obj)
;;; "\003\003_offset "
(vla-put-Macro obj "\003\003mycommand ")
```

8.2.347. Свойство *MainDictionary*

Объект: **PreferencesFiles**.

Описание: главный словарь для проверки орфографии. Возможные значения: строка (имя словаря). Не проверяет наличие устанавливаемого словаря.

```
(setq obj (vla-get-Files
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesFiles2 0b151960>
(vla-get-MainDictionary obj)
;;; "enu"
(vla-put-MainDictionary obj "rus")
```

8.2.348. Свойство *MajorAxis*

Объект: **Ellipse**.

Описание: вектор, определяющий большую полуось эллипса (от центра). Возможные значения: вариант (безопасный массив), содержащий три координаты.

Примечание

Это свойство определяет направление главной оси (с соответствующей корректировкой направления малой оси) и пропорции эллипса.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadEllipse 0b150ba4>
(setq temp (vla-get-MajorAxis obj))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (-228.963 0.0 0.0)
(vla-put-MajorAxis obj (vlax-3d-point '(-200.0 200.0 0.0)))
```

8.2.349. Свойство *MajorRadius*

Объект: **Ellipse**.

Описание: размер большой полуоси. Возможные значения: вещественное число.

Примечание

Это свойство определяет размер большой полуоси без корректировки малой полуоси. Большая полуось не может быть меньше малой. См. свойство `MinorRadius`.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadEllipse 0b150ba4>
(vla-get-MajorRadius obj)
;;; 282.843
(vla-put-MajorRadius obj 450)
```

8.2.350. Свойство *Mask*

Объект: **`LayerStateManager`**.

Описание: битовая маска сохраняемых свойств слоев. Возможные значения (установленные битовые флаги, кроме `acLsAll`, суммируются):

- `acLsNone` (0) — ничего;
- `acLsOn` (1) — включение;
- `acLsFrozen` (2) — заморозка;
- `acLsLocked` (4) — блокировка;
- `acLsPlot` (8) — вывод на печать;
- `acLsNewViewport` (16) — заморозка в новых видовых экранах листа;
- `acLsColor` (32) — цвет;
- `acLsLineType` (64) — тип линий;
- `acLsLineWeight` (128) — вес линий;
- `acLsPlotStyle` (256) — стиль печати;
- `acLsAll` (65535) — все.

Примечание

Объект требует отдельной инициализации после получения указателя на него.

```
(setq obj (vla-GetInterfaceObject
            (vlax-get-acad-object) "AutoCAD.AcadLayerStateManager.16"))
;;; #<VLA-OBJECT IAcadLayerStateManager 08fd2544>
(vla-SetDatabase obj (vla-get-Database actdoc))
;;; инициализация объекта
(vla-get-Mask obj "MyState")
;;; 495
(vla-put-Mask obj "MyState" (+ acLsOn acLsFrozen acLsLocked))
```

8.2.351. Свойство *MaxActiveViewports*

Объект: **DatabasePreferences**.

Описание: максимальное количество активных плавающих видовых экранов. Соответствует системной переменной MAXACTVP. Возможные значения: целое число от 2 до 64, при установке через ActiveX — до 48.

```
(setq obj (vla-get-Preferences actdoc))
;;; #<VLA-OBJECT IAcadDatabasePreferences 05db2f54>
(vla-get-MaxActiveViewports obj)
;;; 64
(vla-put-MaxActiveViewports obj 15)
```

8.2.352. Свойство *MaxAutoCADWindow*

Объект: **PreferencesDisplay**.

Описание: развертывание окна AutoCAD на весь экран при старте. Возможные значения:

- `:vlax-true` — включено;
- `:vlax-false` — выключено.

```
(setq obj (vla-get-Display
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesDisplay 0e69784c>
(vla-get-MaxAutoCADWindow obj)
;;; :vlax-false
(vla-put-MaxAutoCADWindow obj :vlax-true)
```

8.2.353. Свойство *MClose*

Объект: **PolygonMesh**.

Описание: замкнутость в М-направлении. Возможные значения:

- `:vlax-true` — да;
- `:vlax-false` — нет.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadPolygonMesh 0e6e3724>
(vla-get-MClose obj)
;;; :vlax-false
(vla-put-MClose obj :vlax-true)
```

8.2.354. Свойство *MDensity*

Объект: **PolygonMesh**.

Описание: плотность сети в М-направлении. Возможные значения: целое число от 2 до 256.

Примечание

Свойство применимо к сетям типа: acQuadSurfaceMesh, acCubicSurfaceMesh, acBezierSurfaceMesh.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadPolygonMesh 0e6e3724>
(vla-put-Type obj acBezierSurfaceMesh)
(vla-get-MDensity obj)
;;; 7
(vla-put-MDensity obj 11)
```

8.2.355. Свойство *Measurement*

Объекты: **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**.

Описание: фактический размер. Только чтение. Возможные значения: вещественное неотрицательное число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRotated 0e6d93e4>
(vla-get-Measurement obj)
;;; 324.792
```

8.2.356. Свойство *MenuBar*

Объект: **Application**.

Описание: указатель строки меню. Только чтение. Возможные значения: VLA-объект.

```
(setq obj (vlax-get-acad-object))
;;; #<VLA-OBJECT IAcadApplication 00b9d604>
(vla-get-MenuBar obj)
;;; #<VLA-OBJECT IAcadMenuBar 0bc5c5f4>
```

8.2.357. Свойство *MenuFile*

Объект: **PreferencesFiles**.

Описание: имя текущего MNU-файла. Возможные значения: строка (имя файла).

Примечание

Указанный файл компилируется сразу при установке данного свойства, если компилированные файлы меню отсутствуют.

```
(setq obj (vla-get-Files
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesFiles2 0bc5dda0>
(vla-get-MenuFile obj)
;;; "C:\\Documents and ... 2005\\enu\\Support\\ACAD"
(vla-put-MenuFile obj "c:\\temp\\mymenu")
```

8.2.358. Свойство *MenuFileName*

Объект: **MenuGroup**.

Описание: имя MNC- или CUI-файла, в котором описана данная группа меню (группа адаптации). Только чтение. Возможные значения: строка.

```
(setq obj (vla-Item (vla-get-MenuGroups (vlax-get-acad-object)) "acad"))
;;; #<VLA-OBJECT IAcadMenuGroup 010cf0a4>
(vla-get-MenuFileName obj)
;;; "C:\\Documents and ... 2005\\enu\\Support\\ACAD.mnc"
```

8.2.359. Свойство *MenuGroups*

Объект: **Application**.

Описание: указатель семейства групп меню (адаптации). Только чтение. Возможные значения: VLA-объект.

```
(setq obj (vlax-get-acad-object))
;;; #<VLA-OBJECT IAcadApplication 00b9d604>
(vla-get-MenuGroups obj)
;;; #<VLA-OBJECT IAcadMenuGroups 010836a8>
```

8.2.360. Свойство *Menus*

Объект: **MenuGroup**.

Описание: указатель семейства `PopupMenus`. Только чтение. Возможные значения: VLA-объект.

```
(setq obj (vla-Item (vla-get-MenuGroups (vlax-get-acad-object)) "acad"))
;;; #<VLA-OBJECT IAcadMenuGroup 010cf0a4>
(vla-get-Menus obj)
;;; #<VLA-OBJECT IAcadPopupMenus 010fcb74>
```

8.2.361. Свойство *MinimumTableHeight*

2005

Объект: **Table**.

Описание: минимальная высота таблицы. Только чтение. Возможные значения: положительное вещественное число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 010f9f14>
(vla-get-MinimumTableHeight obj)
;;; 36.5
```

8.2.362. Свойство *MinimumTableWidth*

2005

Объект: **Table**.

Описание: минимальная ширина таблицы. Только чтение. Возможные значения: положительное вещественное число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 010f9f14>
(vla-get-MinimumTableWidth obj)
;;; 80.0
```

8.2.363. Свойство *MinorAxis*

Объект: **Ellipse**.

Описание: вектор, определяющий малую полуось эллипса. Только чтение. Возможные значения: вариант (безопасный массив), содержащий три координаты.

Примечание

Для пропорционального изменения размера и/или ориентации эллипса используется свойство `MajorAxis`.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadEllipse 010f8ae4>
(setq temp (vla-get-MinorAxis obj))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (65.0626 -135.822 0.0)
```

8.2.364. Свойство *MinorRadius*

Объект: **Ellipse**.

Описание: размер малой полуоси. Возможные значения: вещественное число.

Примечание

Это свойство определяет размер малой оси без корректировки большой оси. Малая ось не может быть больше большой. См. свойство `MajorRadius`.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadEllipse 010f8ae4>
(vla-get-MinorRadius obj)
;;; 150.601
(vla-put-MinorRadius obj 200)
```

8.2.365. Свойство *MLineScale*

Объект: **`MLine`**.

Описание: масштаб. Возможные значения: вещественное число, может быть отрицательным.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadMLine2 027db2b4>
(vla-get-MLineScale obj)
;;; 20.0
(vla-put-MLineScale obj 10)
```

8.2.366. Свойство *Mode*

Объект: **`Attribute`**.

Описание: битовая маска режимов атрибута в описании. Соответствует системной переменной AFLAGS. Возможные значения (установленные битовые флаги могут суммироваться):

- `acAttributeModeNormal` (0) — обычный;
- `acAttributeModeInvisible` (1) — невидимый;
- `acAttributeModeConstant` (2) — постоянный;
- `acAttributeModeVerify` (4) — проверяемый;
- `acAttributeModePreset` (8) — предустановленный.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadAttribute 054cd534>
(vla-get-Mode obj)
;;; 8
(vla-put-Mode obj (+ acAttributeModeInvisible acAttributeModeConstant))
```

8.2.367. Свойство *ModelCrosshairColor*

Объект: **PreferencesDisplay**.

Описание: цвет перекрестия и названия вкладки в пространстве модели. Возможные значения: вариант типа OLE_COLOR.

Примечание

Этот тип данных не поддерживается в AutoLISP. Тем не менее управлять цветом можно, зная цифровые коды цветов, но нельзя получить значение фактически установленного цвета.

```
(setq obj (vla-get-Display
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesDisplay 0e58592c>
(setq temp (vla-get-ModelCrosshairColor obj))
;;; #<variant 19 16777215>
(vlax-variant-value temp)
;;; error: LISPFromPtr failed. The type is not supported: 19
(vla-put-ModelCrosshairColor obj (vlax-make-variant 12634758 19))
```

8.2.368. Свойство *ModelSpace*

Объект: **Document**.

Описание: указатель на пространство модели. Возможные значения: VLA-объект.

```
(vla-get-ModelSpace actdoc)
;;; #<VLA-OBJECT IAcadModelSpace2 0e5befa4>
```

8.2.369. Свойство *ModelType*

Объекты: **Layout**, **PlotConfiguration**.

Описание: тип вкладки (модели или листа), с которой связана данная конфигурация печати, или тип самой вкладки. Только чтение. Возможные значения:

- :vlax-true — конфигурация печати применяется только для пространства модели или вкладка является пространством модели;
- :vlax-false — конфигурация печати применяется для всех вкладок или вкладка относится к пространству листа.

```
(setq obj (vla-get-ActiveLayout actdoc))
;;; #<VLA-OBJECT IAcadLayout 0e5a1dd4>
(vla-get-ModelType obj)
;;; :vlax-false
```

8.2.370. Свойство *ModelView*

2005

Объект: **PViewport**.

Описание: ассоциированный вид (в подшивках). Возможные значения: VLA-объект вида.

Примечание

Не документировано. Может возвращать `nil`.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadPViewport2 1a0ac0b4>
(vla-get-ModelView obj)
;;; #<VLA-OBJECT IAcadView2 1a0a9194>
(vla-put-ModelView obj (vla-Item (vla-get-Views actdoc) "section 7"))
;;; error: Unknown exception occurred
;;; warning: unwind skipped on unknown exception
```

8.2.371. Свойство *MomentOfInertia*

Объекты: **3DSolid**, **Region**.

Описание: момент инерции. Только чтение. Возможные значения: вариант (безопасный массив), содержащий значения момента инерции относительно двух (для `Region`) или трех (для `3DSolid`) координатных осей.

Примечание

Для получения этого свойства у объекта `Region` он (объект) должен находиться в плане текущей системы координат.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadRegion 052b4c14>
(setq temp (vla-get-MomentOfInertia obj))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (1.632e+008 3.25159e+010)
```

8.2.372. Свойство *MRUNumber*

Объект: **PreferencesOpenSave**.

Описание: число последних открывавшихся файлов, имена которых выводятся в меню **File** (Файл). Только чтение. Возможные значения: целое число от 0 до 9.

```
(setq obj (vla-get-OpenSave
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesOpenSave 0ac6e044>
(vla-get-MRUNumber obj)
;;; 9
```

8.2.373. Свойство *MSpace*

Объект: **Document**.

Описание: признак активизации режима **MODEL** (МОДЕЛЬ) для видового экрана в листе. Возможные значения:

- `:vlax-true` — активен режим **MODEL** (МОДЕЛЬ);
- `:vlax-false` — активен режим **PAPER** (ЛИСТ).

Примечание

Свойство недоступно на вкладке модели.

```
(vla-get-MSpace actdoc)
;;; :vlax-false
(vla-put-MSpace actdoc :vlax-true)
```

8.2.374. Свойство *MTextObjectId*

Объект: **Label**.

Описание: ID текстового объекта ярлыка. Только чтение. Возможные значения: VLA-объект.

```
(setq DBConn (vla-GetInterfaceObject (vlax-get-acad-object)
               "CAO.DbConnect.16"))
;;; #<VLA-OBJECT IDbConnect 0573c770>
(setq DBLinksT (vlax-invoke DBConn 'GetLinkTemplates))
;;; #<VLA-OBJECT ILinkTemplates 0573bbf0>
(setq DBLinkT (vla-Item DBLinksT "Link2"))
;;; #<VLA-OBJECT ILinkTemplate 0573d200>
(setq Links (vlax-invoke-method DBConn 'GetLinks DBLinkT
  (vlax-make-variant (vlax-safearray-fill (vlax-make-safearray
    vlax-vbLong '(0 . 0)) (list (vla-get-ObjectID (vlax-ename->vla-object
      (car (entsel))))))) 2 actdoc))
;;; #<VLA-OBJECT ILinks 05739be0>
(setq Link (vla-Item Links 1))
;;; #<VLA-OBJECT ILink 0573eaa0>
(setq obj (vla-get-Label Link))
;;; #<VLA-OBJECT ILabel 0573e050>
(vlax-get-property obj 'MTextObjectId)
;;; 2130124464
```

8.2.375. Свойство *MVertexCount*

Объект: **PolygonMesh**.

Описание: количество вершин в М-направлении. Возможные значения: целое число от 2 до 256.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadPolygonMesh 0ac871a4>
(vla-get-MVertexCount obj)
;;; 16
```

8.2.376. Свойство *Name*

Объекты: **Application**, **Block**, **BlockRef**, **Dictionary**, **DimStyle**, **Document**, **ExternalReference**, **Group**, **LabelTemplate**, **Layer**, **Layout**, **Linetype**, **LinkTemplate**, **MenuGroup**, **MInsertBlock**, **ModelSpace**, **PaperSpace**, **PlotConfiguration**, **PopupMenu**, **Query**, **Raster**, **RegisteredApplication**, **SelectionSet**, **Shape**, **TextStyle**, **Toolbar**, **ToolbarItem**, **UCS**, **View**, **Viewport**, **XRecord**.

Описание: имя. Для `Application`, `Document`, `MenuGroup`, `ModelSpace`, `PaperSpace`, `SelectionSet` и `TextStyle` — только чтение. Возможные значения: непустая строка.

Примечание

Для вхождения блока изменение данного свойства соответствует замене вставки одного блока на другой. Блок должен быть определен.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadBlockReference 0ac87104>
(vla-get-Name obj)
;;; "TestBlock"
(vla-put-Name obj "Other Block")
```

8.2.377. Свойство *NameNoMnemonic*

Объект: **PopupMenu**.

Описание: название падающего меню без служебных символов мнемоники. Только чтение. Возможные значения: строка.

```
(setq obj (vla-Item (vla-get-Menus (vla-Item (vla-get-MenuGroups
    (vlax-get-acad-object)) "acad")) 9))
;;; #<VLA-OBJECT IAcadPopupMenu 010c9a08>
(vla-get-NameNoMnemonic obj)
;;; "Modify"
```

8.2.378. Свойство *NClose*

Объект: **PolygonMesh**.

Описание: замкнутость в N-направлении. Возможные значения:

- `:vlax-true` — да;
- `:vlax-false` — нет.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadPolygonMesh 0ac8d014>
(vla-get-NClose obj)
;;; :vlax-true
(vla-put-NClose obj :vlax-false)
```

8.2.379. Свойство *NDensity*

Объект: **PolygonMesh**.

Описание: плотность сети в N-направлении. Возможные значения: целое число от 2 до 256.

Примечание

Свойство применимо к сетям типа acQuadSurfaceMesh, acCubicSurfaceMesh, acBezierSurfaceMesh.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadPolygonMesh 0ac8d014>
(vla-put-Type obj acBezierSurfaceMesh)
(vla-get-NDensity obj)
;;; 7
(vla-put-NDensity obj 11)
```

8.2.380. Свойство *Normal*

Объекты: **Arc**, **Attribute**, **AttributeReference**, **BlockRef**, **Circle**, **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**, **Ellipse**, **ExternalReference**, **Hatch**, **Leader**, **LightweightPolyline**, **Line**, **MInsertBlock**, **MText**, **Point**, **Polyline**, **Region**, **Shape**, **Solid**, **Text**, **Tolerance**, **Trace**.

Описание: единичный вектор нормали к плоскости построения примитива в мировой системе координат. Для **Region** — только чтение. Возможные значения: вариант (безопасный массив), содержащий три координаты вектора.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadEllipse 0ac9b8b4>
(setq temp (vla-get-Normal obj))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (0.0 0.0 1.0)
(vla-put-Normal obj (vlax-3d-point '(-1.0 0.0 -1.0)))
```

8.2.381. Свойство *NumberOfControlPoints*

Объект: **Spline**.

Описание: количество управляющих точек. Только чтение. Возможные значения: положительное целое число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadSpline 0afe3924>
(vla-get-NumberOfControlPoints obj)
;;; 6
```

8.2.382. Свойство *NumberOfCopies*

Объект: **Plot**.

Описание: число копий, выводимых на печать. Возможные значения: положительное целое число.

```
(setq obj (vla-get-Plot actdoc))
;;; #<VLA-OBJECT IAcadPlot 0b008f9c>
(vla-get-NumberOfCopies obj)
;;; 1
(vla-put-NumberOfCopies obj 3)
```

8.2.383. Свойство *NumberOfFaces*

Объект: **PolyFaceMesh**.

Описание: количество граней. Только чтение. Возможные значения: неотрицательное целое число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadPolyfaceMesh 142ec974>
(vla-get-NumberOfFaces obj)
;;; 1
```

8.2.384. Свойство *NumberOfFitPoints*

Объект: **Spline**.

Описание: количество определяющих точек. Только чтение. Возможные значения: положительное целое число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadSpline 0afe3924>
(vla-get-NumberOfFitPoints obj)
;;; 4
```

8.2.385. Свойство *NumberOfLoops*

Объект: **Hatch**.

Описание: число циклов в границе. Только чтение. Возможные значения: положительное целое число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadHatch 0b00ecf4>
(vla-get-NumberOfLoops obj)
;;; 1
```

8.2.386. Свойство *NumberOfVertices*

Объект: **PolyFaceMesh**.

Описание: количество вершин. Только чтение. Возможные значения: положительное целое число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadPolyfaceMesh 0affe684>
(vla-get-NumberOfVertices obj)
;;; 4
```

8.2.387. Свойство *NumericScale*

Объект: **KeyDescription**.

Описание: числовой масштаб поля данных. Возможные значения: неотрицательное целое число.

Примечание

Актуально только для числового типа данных.

```
(setq DBConn (vla-GetInterfaceObject (vlax-get-acad-object)
               "CAO.DbConnect.16"))
;;; #<VLA-OBJECT IDbConnect 0573c770>
(setq DBLinksT (vlax-invoke DBConn 'GetLinkTemplates))
;;; #<VLA-OBJECT ILinkTemplates 05739c90>
(setq DBLinkT (vla-Item DBLinksT 0)) ;_был создан заранее
;;; #<VLA-OBJECT ILinkTemplate 0573d380>
(setq KeyDeskrs (vlax-get-property DBLinkT 'KeyDescriptions))
;;; #<VLA-OBJECT IKeyDescriptions 05744e70>
(setq KeyDeskr (vla-Item KeyDeskrs 0))
;;; #<VLA-OBJECT IKeyDescription 05743e70>
(vlax-get-property KeyDeskr 'NumericScale)
;;; 0
(vlax-put-property KeyDeskr 'NumericScale 3)
```

8.2.388. Свойство *NVertexCount*

Объект: **PolygonMesh**.

Описание: количество вершин в N-направлении. Возможные значения: целое число от 2 до 256.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadPolygonMesh 0b0575b4>
(vla-get-NVertexCount obj)
;;; 16
```

8.2.389. Свойство *ObjectID*

Объекты: **все графические объекты**, **Block**, **Blocks**, **Dictionary**, **Dictionaries**, **DimStyle**, **DimStyles**, **Group**, **Groups**, **Layer**, **Layers**, **Layout**, **Layouts**, **Linetype**, **Linetypes**, **Link**, **ModelSpace**, **PaperSpace**, **PlotConfiguration**, **PlotConfigurations**, **RegisteredApplication**, **RegisteredApplications**, **TextStyle**, **TextStyles**, **UCS**, **UCSs**, **View**, **Views**, **Viewport**, **Viewports**, **XRecord**.

Описание: уникальный, в пределах области определения, идентификатор (ID) объекта. Только чтение. Возможные значения: положительное целое число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadLWPolyline 0b049144>
(vla-get-ObjectID obj)
;;; 2130034608
```

8.2.390. Свойство *ObjectName*

Объекты: **все графические объекты**, **Block**, **Blocks**, **Dictionary**, **Dictionaries**, **DimStyle**, **DimStyles**, **Group**, **Groups**, **Layer**, **Layers**, **Layout**, **Layouts**, **Linetype**, **Linetypes**, **ModelSpace**, **PaperSpace**, **PlotConfiguration**, **PlotConfigurations**, **RegisteredApplication**, **RegisteredApplications**, **TextStyle**, **TextStyles**, **UCS**, **UCSs**, **View**, **Views**, **Viewport**, **Viewports**, **XRecord**.

Описание: тип (имя класса) объекта. Только чтение. Возможные значения: строка.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadLWPolyline 0b049144>
(vla-get-ObjectName obj)
;;; "AcDbPolyline"
```

8.2.391. Свойство *ObjectSnapMode*

Объект: **Document**.

Описание: состояние режима **OSNAP** (ПРИВЯЗКА).

Возможные значения:

- `:vlax-true` — включен;
- `:vlax-false` — выключен.

```
(vla-get-ObjectSnapMode actdoc)
;;; :vlax-true
(vla-put-ObjectSnapMode actdoc :vlax-false)
```

8.2.392. Свойство *ObjectSortByPlotting*

Объект: **DatabasePreferences**.

Описание: сортировка объектов при печати. Соответствует системной переменной SORTENTS. Возможные значения:

- `:vlax-true` — включена;
- `:vlax-false` — выключена.

```
(setq obj (vla-get-Preferences actdoc))
;;; #<VLA-OBJECT IAcadDatabasePreferences 0b056644>
(vla-get-ObjectSortByPlotting obj)
;;; :vlax-true
(vla-put-ObjectSortByPlotting obj :vlax-false)
```

8.2.393. Свойство *ObjectSortByPSOutput*

Объект: **DatabasePreferences**.

Описание: сортировка объектов при выводе в PS-файлы. Соответствует системной переменной SORTENTS. Возможные значения:

- `:vlax-true` — включена;
- `:vlax-false` — выключена.

```
(setq obj (vla-get-Preferences actdoc))
;;; #<VLA-OBJECT IAcadDatabasePreferences 0b056644>
(vla-get-ObjectSortByPSOutput obj)
;;; :vlax-true
(vla-put-ObjectSortByPSOutput obj :vlax-false)
```

8.2.394. Свойство *ObjectSortByRedraws*

Объект: **DatabasePreferences**.

Описание: сортировка объектов при освежении экрана. Соответствует системной переменной SORTENTS.

Возможные значения:

- `:vlax-true` — включена;
- `:vlax-false` — выключена.

```
(setq obj (vla-get-Preferences actdoc))
;;; #<VLA-OBJECT IAcadDatabasePreferences 0b056644>
(vla-get-ObjectSortByRedraws obj)
;;; :vlax-true
(vla-put-ObjectSortByRedraws obj :vlax-false)
```

8.2.395. Свойство *ObjectSortByRegens*

Объект: **DatabasePreferences**.

Описание: сортировка объектов при регенерации экрана. Соответствует системной переменной SORTENTS. Возможные значения:

- `:vlax-true` — включена;
- `:vlax-false` — выключена.

```
(setq obj (vla-get-Preferences actdoc))
;;; #<VLA-OBJECT IAcadDatabasePreferences 0b056644>
(vla-get-ObjectSortByRegens obj)
;;; :vlax-true
(vla-put-ObjectSortByRegens obj :vlax-false)
```

8.2.396. Свойство *ObjectSortBySelection*

Объект: **DatabasePreferences**.

Описание: сортировка объектов при операции выбора объектов. Соответствует системной переменной SORTENTS. Возможные значения:

- `:vlax-true` — включена;
- `:vlax-false` — выключена.

```
(setq obj (vla-get-Preferences actdoc))
;;; #<VLA-OBJECT IAcadDatabasePreferences 0b056644>
(vla-get-ObjectSortBySelection obj)
;;; :vlax-true
(vla-put-ObjectSortBySelection obj :vlax-false)
```

8.2.397. Свойство *ObjectSortBySnap*

Объект: **DatabasePreferences**.

Описание: сортировка объектов при объектной привязке. Соответствует системной переменной SORTENTS.

Возможные значения:

- `:vlax-true` — включена;
- `:vlax-false` — выключена.

```
(setq obj (vla-get-Preferences actdoc))
;;; #<VLA-OBJECT IAcadDatabasePreferences 0b056644>
(vla-get-ObjectSortBySnap obj)
;;; :vlax-true
(vla-put-ObjectSortBySnap obj :vlax-false)
```

8.2.398. Свойство *ObliqueAngle*

Объекты: **Attribute**, **AttributeReference**, **Shape**, **Text**, **TextStyle**.

Описание: угол наклона букв. Возможные значения: вещественное число.

Примечание

Угол наклона задается в радианах и должен попадать в диапазон ±1.48352986419518 (±85º). Положительное значение соответствует наклону вправо, отрицательное — влево. Выполняется автоматическое приведение угла к интервалу от 0 до 2π.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadText2 0b048c84>
(vla-get-ObliqueAngle obj)
;;; 0.0
(vla-put-ObliqueAngle obj 0.2)
```

8.2.399. Свойство *OleItemType*

Объект: **Ole**.

Описание: тип OLE-объекта. Возможные значения:

- `acOTLink` (1) — связанный;
- `acOTEmbedded` (2) — внедренный;
- `acOTStatic` (3) — статический.

Примечание

Не документировано. Установка недопустимых значений игнорируется без возникновения ошибки.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadOle 0b2e1644>
(vla-get-OleItemType obj)
;;; 2
(vla-put-OleItemType obj 3)
```

8.2.400. Свойство *OLELaunch*

Объект: **DatabasePreferences**.

Описание: признак запуска родительского приложения при печати OLE-объектов. Соответствует системной переменной OLESTARTUP. Возможные значения:

- `:vlax-true` — да;
- `:vlax-false` — нет.

```
(setq obj (vla-get-Preferences actdoc))
;;; #<VLA-OBJECT IAcadDatabasePreferences 0b2d87b4>
(vla-get-OLELaunch obj)
;;; :vlax-false
(vla-put-OLELaunch obj :vlax-true)
```

8.2.401. Свойство *OlePlotQuality*

Объект: **Ole**.

Описание: индивидуальное качество печати OLE-объекта. Возможные значения:

- `acOPQMonochrome` (0) — монохромное;
- `acOPQLowGraphics` (1) — графическое с низким разрешением;
- `acOPQHighGraphics` (2) — графическое с высоким разрешением.

Примечание

Не документировано. Установка недопустимых значений приводит к автоматической установке 1, без возникновения ошибки.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadOle 0b2e1644>
(vla-get-OlePlotQuality obj)
;;; 1
(vla-put-OlePlotQuality obj 2)
```

8.2.402. Свойство *OLEQuality*

Объект: **PreferencesOutput**.

Описание: общая настройка качества печати OLE-объектов. Соответствует системной переменной OLEQUALITY. Возможные значения:

- `acOQLineArt` (0) — штриховой рисунок;
- `acOQText` (1) — текст;

- `acOQGraphics` (2) — деловая графика;
- `acOQPhoto` (3) — фото;
- `acOQHighPhoto` (4) — высококачественное фото.

```
(setq obj (vla-get-Output
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesOutput2 0b30a428>
(vla-get-OLEQuality obj)
;;; 2
(vla-put-OLEQuality obj acOQPhoto)
```

8.2.403. Свойство *OleSourceApp*

2005

Объект: `Ole`.

Описание: название типа документа по родительскому приложению. Возможные значения: строка.

Примечание

Не документировано. Установка недопустимых значений игнорируется без возникновения ошибки.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadOle 0b304864>
(vla-get-OleSourceApp obj)
;;; "Microsoft Word Document"
(vla-put-OleSourceApp obj "Any other document")
```

8.2.404. Свойство *OnMenuBar*

Объект: `PopupMenu`.

Описание: признак присутствия заголовка падающего меню в строке меню. Только чтение. Возможные значения:

- `:vlax-true` — да;
- `:vlax-false` — нет.

```
(setq obj (vla-Item (vla-get-Menus (vla-Item
    (vla-get-MenuGroups (vlax-get-acad-object)) "acad")) 9))
;;; #<VLA-OBJECT IAcadPopupMenu 010aec98>
(vla-get-OnMenuBar obj)
;;; :vlax-true
```

8.2.405. Свойство *OpenSave*

Объект: **Preferences**.

Описание: указатель на объект, содержащий настройки файловых операций. Только чтение. Возможные значения: VLA-объект.

```
(setq obj (vla-get-Preferences (vlax-get-acad-object)))
;;; #<VLA-OBJECT IAcadPreferences 0b033e8c>
(vla-get-OpenSave obj)
;;; #<VLA-OBJECT IAcadPreferencesOpenSave 0b033e74>
```

8.2.406. Свойство *Origin*

Объекты: **Block**, **Layout**, **PlotConfiguration**, **Raster**, **UCS**.

Описание: базовая точка вставки в мировой системе координат. Возможные значения: координаты трехмерной точки в виде варианта с безопасным массивом.

```
(setq obj (vla-Item (vla-get-Blocks actdoc) 3))
;;; #<VLA-OBJECT IAcadBlock2 0905b994>
(setq temp (vla-get-Origin obj))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (0.0 0.0 0.0)
(vla-put-Origin obj (vlax-3d-point (getpoint)))
```

8.2.407. Свойство *OrthoOn*

Объект: **Viewport**.

Описание: состояние режима **ORTHO** (ОРТО). Возможные значения:

- `:vlax-true` — включен;
- `:vlax-false` — выключен.

```
(setq obj (vla-Item (vla-get-Viewports actdoc) 0))
;;; #<VLA-OBJECT IAcadViewport 0b033354>
(vla-get-OrthoOn obj)
;;; :vlax-false
(vla-put-OrthoOn obj :vlax-true)
```

8.2.408. Свойство *Output*

Объект: **Preferences**.

Описание: указатель на объект, содержащий настройки операций вывода. Только чтение. Возможные значения: VLA-объект.

```
(setq obj (vla-get-Preferences (vlax-get-acad-object)))
;;; #<VLA-OBJECT IAcadPreferences 0b033e8c>
(vla-get-Output obj)
;;; #<VLA-OBJECT IAcadPreferencesOutput2 0b032818>
```

8.2.409. Свойство *OverrideCenter*

2006

Объект: **DimRadialLarge**.

Описание: переопределенная точка центра. Возможные значения: координаты трехмерной точки в виде варианта с безопасным массивом.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRadialLarge 027f3724>
(setq temp (vla-get-OverrideCenter obj))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (297.317 189.571 0.0)
(vla-put-OverrideCenter obj (vlax-3d-point (getpoint)))
```

8.2.410. Свойство *OwnerID*

Объекты: **все графические объекты**, **Block**, **Blocks**, **Dictionary**, **Dictionaries**, **DimStyle**, **DimStyles**, **Group**, **Groups**, **Layer**, **Layers**, **Layout**, **Layouts**, **Linetype**, **Linetypes**, **ModelSpace**, **PaperSpace**, **PlotConfiguration**, **PlotConfigurations**, **RegisteredApplication**, **RegisteredApplications**, **TextStyle**, **TextStyles**, **UCS**, **UCSs**, **View**, **Views**, **Viewport**, **Viewports**, **XRecord**.

Описание: ID владельца. Только чтение. Возможные значения: целое число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadCircle 0b0338b4>
(vla-get-OwnerID obj)
;;; 2130029848
```

8.2.411. Свойство *PageSetupOverridesTemplateFile*

2005

Объект: **PreferencesFiles**.

Описание: шаблон для настроек листов, по умолчанию. Возможные значения: строка (имя файла шаблона). Не проверяет наличие устанавливаемого шаблона.

```
(setq obj (vla-get-Files
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesFiles2 0b035f40>
```

```
(vla-get-PageSetupOverridesTemplateFile obj)
;;; "C:\\Documents and Settings\\All ... Imperial.dwt"
(vla-put-PageSetupOverridesTemplateFile obj "Any String")
```

8.2.412. Свойство *PaperSpace*

Объект: **Document**.

Описание: указатель на текущую вкладку пространства листа. Возможные значения: VLA-объект.

```
(vla-get-PaperSpace actdoc)
;;; #<VLA-OBJECT IAcadPaperSpace2 0b032614>
```

8.2.413. Свойство *PaperUnits*

Объекты: **Layout**, **PlotConfiguration**.

Описание: единицы измерения, используемые при показе свойств листа и конфигурации печати. Возможные значения:

- `acInches` (0) — дюймы;
- `acMillimeters` (1) — миллиметры;
- `acPixels` (2) — пикселы.

Примечание

Изменение возможно только на совместимый тип (дюймы/миллиметры).

```
(setq obj (vla-get-ActiveLayout actdoc))
;;; #<VLA-OBJECT IAcadLayout 0b037824>
(vla-get-PaperUnits obj)
;;; 1
(vla-put-PaperUnits obj acInches)
```

8.2.414. Свойство *Parent*

Объекты: **MenuBar**, **MenuGroup**, **MenuGroups**, **PopupMenu**, **PopupMenus**, **PopupMenuItem**, **Toolbar**, **Toolbars**, **ToolbarItem**.

Описание: указатель родительского объекта. Только чтение. Возможные значения: VLA-объект.

```
(setq obj (vla-get-MenuGroups (vlax-get-acad-object)))
;;; #<VLA-OBJECT IAcadMenuGroups 010836a8>
(vla-get-Parent obj)
;;; #<VLA-OBJECT IAcadApplication 00b9d604>
```

8.2.415. Свойство *Password*

Объект: **SecurityParams**.

Описание: пароль. Возможные значения: строка, в том числе пустая.

```
(setq obj (vla-GetInterfaceObject
            (vlax-get-acad-object) "AutoCAD.SecurityParams.16"))
;;; #<VLA-OBJECT IAcadSecurityParams 0b04b864>
(vla-get-Password obj)
;;; ""
(vla-put-Password obj "My topsecret pass")
```

8.2.416. Свойство *Path*

Объекты: **Application**, **Block**, **Document**, **ExternalReference**.

Описание: путь к файлу. Возможные значения: строка, не проверяется.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadExternalReference 127639f4>
(vla-get-Path obj)
;;; "D:\\Tutor\\Draw\\Other.dwg"
(vla-put-Path obj "Any string")
```

8.2.417. Свойство *PatternAngle*

Объект: **Hatch**.

Описание: угол наклона штриховки в радианах. Переопределяет значение, установленное системной переменной HPANG. Возможные значения: от 0 до 2π.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadHatch 12761794>
(vla-get-PatternAngle obj)
;;; 0.0
(vla-put-PatternAngle obj (/ pi 4))
(vla-Evaluate obj)
(vla-update obj)
```

8.2.418. Свойство *PatternDouble*

Объект: **Hatch**.

Описание: признак штриховки "крест-накрест". Переопределяет значение, установленное системной переменной HPDOUBLE. Возможные значения:

- `:vlax-true` — да;
- `:vlax-false` — нет.

Примечание

Не актуально для штриховок типа acHatchPatternTypePreDefined или acHatchPatternTypeCustomDefined, см. свойство PatternType.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadHatch 0b096014>
(vla-get-PatternDouble obj)
;;; :vlax-false
(vla-put-PatternDouble obj :vlax-true)
```

8.2.419. Свойство *PatternName*

Объект: **Hatch**.

Описание: имя образца штриховки. Только чтение. Устанавливается системной переменной HPNAME. Возможные значения: строка.

Примечание

Документация утверждает, что свойство можно изменять — но это не так.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadHatch 12761794>
(vla-get-PatternName obj)
;;; "ANGLE"
```

8.2.420. Свойство *PatternScale*

Объект: **Hatch**.

Описание: масштаб штриховки. Переопределяет значение, установленное системной переменной HPSCALE. Возможные значения: положительное вещественное число.

Примечание

Не актуально для штриховок типа acHatchPatternTypeUserDefined, см. свойство PatternType.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadHatch 12761794>
(vla-get-PatternScale obj)
;;; 0.25
(vla-put-PatternScale obj 0.75)
(vla-Evaluate obj)
(vla-Update obj)
```

8.2.421. Свойство *PatternSpace*

Объект: **Hatch**.

Описание: шаг линий пользовательской штриховки. Переопределяет значение, установленное системной переменной HPSPACE. Возможные значения: положительное вещественное число.

Примечание

Не актуально для штриховок типа `acHatchPatternTypePreDefined` или `acHatchPatternTypeCustomDefined`, см. свойство `PatternType`.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadHatch 0b096014>
(vla-get-PatternSpace obj)
;;; 5.0
(vla-put-PatternSpace obj 7.25)
```

8.2.422. Свойство *PatternType*

Объект: **Hatch**.

Описание: тип штриховки. Возможные значения:

- `acHatchPatternTypeUserDefined` (0) — пользовательская;
- `acHatchPatternTypePreDefined` (1) — предопределенная в файле acad.pat;
- `acHatchPatternTypeCustomDefined` (2) — определенная в ином файле.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadHatch 0b091d64>
(vla-get-PatternType obj)
;;; 1
```

8.2.423. Свойство *Perimeter*

Объект: **Region**.

Описание: периметр. Только чтение. Возможные значения: положительное вещественное число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadRegion 0ae3e9d4>
(vla-get-Perimeter obj)
;;; 263.251
```

8.2.424. Свойство *PickAdd*

Объект: **PreferencesSelection**.

Описание: режим добавления в набор объектов только с помощью клавиши <Shift>. Соответствует системной переменной PICKADD. Возможные значения:

- `:vlax-true` — включено;
- `:vlax-false` — выключено.

```
(setq obj (vla-get-Selection
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesSelection 0ae3ef90>
(vla-get-PickAdd obj)
;;; :vlax-true
(vla-put-PickAdd obj :vlax-false)
```

8.2.425. Свойство *PickAuto*

Объект: **PreferencesSelection**.

Описание: автоматический переход на выбор рамкой/секрамкой при промахе во время выбора. Соответствует системной переменной PICKAUTO. Возможные значения:

- `:vlax-true` — включен;
- `:vlax-false` — выключен.

```
(setq obj (vla-get-Selection
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesSelection 0ae3ef90>
(vla-get-PickAuto obj)
;;; :vlax-true
(vla-put-PickAuto obj :vlax-false)
```

8.2.426. Свойство *PickBoxSize*

Объект: **PreferencesSelection**.

Описание: размер прицела выбора объектов. Соответствует системной переменной PICKBOX. Возможные значения: целое число от 0 до 50.

```
(setq obj (vla-get-Selection
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesSelection 0ae3ef90>
(vla-get-PickBoxSize obj)
;;; 4
(vla-put-PickBoxSize obj 6)
```

8.2.427. Свойство *PickDrag*

Объект: **PreferencesSelection**.

Описание: выбор рамкой/секрамкой только при удержании левой кнопки мыши. Соответствует системной переменной PICKDRAG. Возможные значения:

- :vlax-true — включен;
- :vlax-false — выключен.

```
(setq obj (vla-get-Selection
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesSelection 0ae3ef90>
(vla-get-PickDrag obj)
;;; :vlax-false
(vla-put-PickDrag obj :vlax-true)
```

8.2.428. Свойство *PickFirst*

Объект: **PreferencesSelection**.

Описание: режим допустимости предварительного выбора. Соответствует системной переменной PICKFIRST. Возможные значения:

- :vlax-true — включен;
- :vlax-false — выключен.

```
(setq obj (vla-get-Selection
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesSelection 0ae3ef90>
(vla-get-PickFirst obj)
;;; :vlax-true
(vla-put-PickFirst obj :vlax-false)
```

8.2.429. Свойство *PickfirstSelectionSet*

Объект: **Document**.

Описание: указатель набора предварительно выбранных объектов. Только чтение. Возможные значения: VLA-объект.

```
(vla-get-PickfirstSelectionSet actdoc)
;;; #<VLA-OBJECT IAcadSelectionSet 0ae39994>
```

8.2.430. Свойство *PickGroup*

Объект: **PreferencesSelection**.

Описание: режим выбора группы указанием любого ее члена.

Возможные значения:

- :vlax-true — включен;
- :vlax-false — выключен.

```
(setq obj (vla-get-Selection
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesSelection 0ae3ef90>
(vla-get-PickGroup obj)
;;; :vlax-true
(vla-put-PickGroup obj :vlax-false)
```

8.2.431. Свойство *Plot*

Объект: **Document**.

Описание: образ печати. Только чтение. Возможные значения: VLA-объект.

```
(vla-get-Plot actdoc)
;;; #<VLA-OBJECT IAcadPlot 0adeec6c>
```

8.2.432. Свойство *PlotConfigurations*

Объект: **Document**.

Описание: семейство конфигураций печати. Только чтение. Возможные значения: VLA-объект.

```
(vla-get-PlotConfigurations actdoc)
;;; #<VLA-OBJECT IAcadPlotConfigurations 0adee3e4>
```

8.2.433. Свойство *PlotHidden*

Объекты: **Layout**, **PlotConfiguration**.

Описание: режим скрытия невидимых линий при печати. Возможные значения:

- :vlax-true — включен;
- :vlax-false — выключен.

```
(setq obj (vla-get-ActiveLayout actdoc))
;;; #<VLA-OBJECT IAcadLayout 0adeea74>
(vla-get-PlotHidden obj)
;;; :vlax-false
(vla-put-PlotHidden obj :vlax-true)
```

8.2.434. Свойство *PlotLegacy*

Объект: **PreferencesOutput**.

Описание: разрешение на использование файлов пакетной печати предыдущих версий. Возможные значения:

- `:vlax-true` — разрешено;
- `:vlax-false` — не разрешено.

```
(setq obj (vla-get-Output
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesOutput2 0adf9718>
(vla-get-PlotLegacy obj)
;;; :vlax-false
(vla-put-PlotLegacy obj :vlax-true)
```

8.2.435. Свойство *PlotLogFilePath* 2005

Объект: **PreferencesFiles**.

Описание: путь к файлу протокола печати. Возможные значения: строка. Не проверяет существование устанавливаемого пути.

```
(setq obj (vla-get-Files
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesFiles2 0adf9230>
(vla-get-PlotLogFilePath obj)
;;; "C:\\Documents and ... enu\\"
(vla-put-PlotLogFilePath obj "K:\\Any Path\\")
```

8.2.436. Свойство *PlotOrigin*

Объекты: **Layout**, **PlotConfiguration**.

Описание: смещение левого нижнего угла зоны печати относительно левого нижнего угла области печати. Возможные значения: координаты двумерной точки в виде варианта с безопасным массивом.

```
(setq obj (vla-get-ActiveLayout actdoc))
;;; #<VLA-OBJECT IAcadLayout 0adeea74>
(setq temp (vla-get-PlotOrigin obj))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (0.0 0.0)
(vla-put-PlotOrigin obj (vlax-safearray-fill
  (vlax-make-safearray vlax-vbDouble '(0 . 1)) '(80 100)))
```

8.2.437. Свойство *PlotPolicy*

Объект: **PreferencesOutput**.

Описание: тип назначения стилей печати по умолчанию. Соответствует системной переменной PSTYLEPOLICY. Возможные значения:

- acPolicyNamed (0) — именованные стили;
- acPolicyLegacy (1) — цветозависимые стили.

```
(setq obj (vla-get-Output
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesOutput2 0adffc68>
(vla-get-PlotPolicy obj)
;;; 1
(vla-put-PlotPolicy obj acPolicyNamed)
```

8.2.438. Свойство *PlotRotation*

Объекты: **Layout**, **PlotConfiguration**.

Описание: поворот области печати. Возможные значения:

- ac0degrees (0) — без поворота;
- ac90degrees (1) — 90°;
- ac180degrees (2) — 180°;
- ac270degrees (3) — 270°.

```
(setq obj (vla-get-ActiveLayout actdoc))
;;; #<VLA-OBJECT IAcadLayout 0adeea74>
(vla-get-PlotRotation obj)
;;; 1
(vla-put-PlotRotation obj ac270degrees)
```

8.2.439. Свойство *PlotStyleName*

Объекты: **все графические объекты**, **Group**, **Layer**.

Описание: имя стиля печати. Для Group — только установка. Возможные значения: непустая строка. Значение не проверяется.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadCircle 0adf9a54>
(vla-get-PlotStyleName obj)
;;; "ByLayer"
(vla-put-PlotStyleName obj "Any String")
```

8.2.440. Свойство *Plottable*

Объект: **Layer**.

Описание: признак печатаемости. Возможные значения:

- :vlax-true — печатаемый;
- :vlax-false — непечатаемый.

```
(setq obj (vla-Item (vla-get-Layers actdoc) "Layer1"))
;;; #<VLA-OBJECT IAcadLayer2 0adff424>
(vla-get-Plottable obj)
;;; :vlax-true
(vla-put-Plottable obj :vlax-false)
```

8.2.441. Свойство *PlotType*

Объекты: **Layout**, **PlotConfiguration**.

Описание: зона печати. Возможные значения:

- acDisplay (0) — экран;
- acExtents (1) — границы;
- acLimits (2) — лимиты;
- acView (3) — вид;
- acWindow (4) — окно;
- acLayout (5) — лист.

Примечание

При установке необходимо учитывать, что некоторые зоны печати могут требовать предварительной установки других свойств.

```
(setq obj (vla-get-ActiveLayout actdoc))
;;; #<VLA-OBJECT IAcadLayout 0adffab4>
(vla-get-PlotType obj)
;;; 0
(vla-put-PlotType obj acExtents)
```

8.2.442. Свойство *PlotViewportBorders*

Объекты: **Layout**, **PlotConfiguration**.

Описание: признак печати границ видовых экранов. Возможные значения:

- :vlax-true — печатать;
- :vlax-false — не печатать.

```
(setq obj (vla-get-ActiveLayout actdoc))
;;; #<VLA-OBJECT IAcadLayout 0adffab4>
(vla-get-PlotViewportBorders obj)
;;; :vlax-false
(vla-put-PlotViewportBorders obj :vlax-true)
```

8.2.443. Свойство *PlotViewportsFirst*

Объекты: **Layout**, **PlotConfiguration**.

Описание: порядок печати видовых экранов и объектов пространства листа. Возможные значения:

- `:vlax-true` — объекты листа в конце;
- `:vlax-false` — объекты листа сначала.

```
(setq obj (vla-get-ActiveLayout actdoc))
;;; #<VLA-OBJECT IAcadLayout 0adffab4>
(vla-get-PlotViewportsFirst obj)
;;; :vlax-true
(vla-put-PlotViewportsFirst obj :vlax-false)
```

8.2.444. Свойство *PlotWithLineweights*

Объекты: **Layout**, **PlotConfiguration**.

Описание: учет весов линий. Возможные значения:

- `:vlax-true` — учитывать;
- `:vlax-false` — игнорировать.

Примечание

Возможно получение комбинации, недостижимой при диалоговой настройке печати — печать с применением стиля печати, но без учета толщин линий.

```
(setq obj (vla-get-ActiveLayout actdoc))
;;; #<VLA-OBJECT IAcadLayout 0ae223b4>
(vla-get-PlotWithLineweights obj)
;;; :vlax-true
(vla-put-PlotWithLineweights obj :vlax-false)
```

8.2.445. Свойство *PlotWithPlotStyles*

Объекты: **Layout**, **PlotConfiguration**.

Описание: использование стилей при печати.

Возможные значения:

- `:vlax-true` — использовать;
- `:vlax-false` — не использовать.

Примечание

Возможно получение комбинации, недостижимой при диалоговой настройке печати — печать с применением стиля печати, но без учета толщин линий.

```
(setq obj (vla-get-ActiveLayout actdoc))
;;; #<VLA-OBJECT IAcadLayout 0ae223b4>
(vla-get-PlotWithPlotStyles obj)
;;; :vlax-true
(vla-put-PlotWithPlotStyles obj :vlax-false)
```

8.2.446. Свойство *PolarTrackingVector*

Объект: **PreferencesDrafting**.

Описание: отображение линий полярного отслеживания. Соответствует системной переменной TRACKPATH. Возможные значения:

- `:vlax-true` — включено;
- `:vlax-false` — выключено.

```
(setq obj (vla-get-Drafting
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesDrafting 0b66d5e8>
(vla-get-PolarTrackingVector obj)
;;; :vlax-true
(vla-put-PolarTrackingVector obj :vlax-false)
```

8.2.447. Свойство *PostScriptPrologFile*

Объект: **PreferencesFiles**.

Описание: имя управляющей секции в файле acad.psf для настройки команды PSOUT (ЭКСПОРТПС). Возможные значения: строка. Не проверяется.

```
(setq obj (vla-get-Files
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesFiles2 0b66d480>
(vla-get-PostScriptPrologFile obj)
;;; ""
(vla-put-PostScriptPrologFile obj "Any string")
```

8.2.448. Свойство *Precision*

Объект: **KeyDescription**.

Описание: точность, количество десятичных знаков. Возможные значения: неотрицательное целое число.

Примечание

Актуально только для числовых (точность данных), текстовых (длина строки символов) и бинарных (размер поля) типов данных.

```
(setq DBConn (vla-GetInterfaceObject (vlax-get-acad-object)
               "CAO.DbConnect.16"))
;;; #<VLA-OBJECT IDbConnect 0573c770>
(setq DBLinksT (vlax-invoke DBConn 'GetLinkTemplates))
;;; #<VLA-OBJECT ILinkTemplates 05739c90>
(setq DBLinkT (vla-Item DBLinksT 0)) ;_был создан заранее
;;; #<VLA-OBJECT ILinkTemplate 0573d380>
(setq KeyDeskrs (vlax-get-property DBLinkT 'KeyDescriptions))
;;; #<VLA-OBJECT IKeyDescriptions 05744e70>
(setq KeyDeskr (vla-Item KeyDeskrs 0))
;;; #<VLA-OBJECT IKeyDescription 05743e70>
(vlax-get-property KeyDeskr 'Precision)
;;; 0
(vlax-put-property KeyDeskr 'Precision 3)
```

8.2.449. Свойство *Preferences*

Объекты: **Application**, **Document**.

Описание: объект настроек. Только чтение. Возможные значения: VLA-объект.

```
(vla-get-Preferences (vlax-get-acad-object))
;;; #<VLA-OBJECT IAcadPreferences 0b66efdc>
(vla-get-Preferences actdoc)
;;; #<VLA-OBJECT IAcadDatabasePreferences 0b66e0f4>
```

8.2.450. Свойство *Preset*

Объекты: **Attribute**, **AttributeReference**.

Описание: свойство "предустановленный". Возможные значения:

- `:vlax-true` — включено;
- `:vlax-false` — выключено.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadAttribute 0b6b3fa4>
(vla-get-Preset obj)
;;; :vlax-false
(vla-put-Preset obj :vlax-true)
```

8.2.451. Свойство *PrimaryUnitsPrecision*

Объекты: **DimAligned**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**, **Tolerance**.

Описание: точность допуска основных единиц примитива. Переопределяет значение, установленное системной переменной DIMDEC. Возможные значения:

- `acDimPrecisionZero` (0) — 0 десятичных знаков;
- `acDimPrecisionOne` (1) — 1 десятичный знак;
- `acDimPrecisionTwo` (2) — 2 десятичных знака;
- `acDimPrecisionThree` (3) — 3 десятичных знака;
- `acDimPrecisionFour` (4) — 4 десятичных знака;
- `acDimPrecisionFive` (5) — 5 десятичных знаков;
- `acDimPrecisionSix` (6) — 6 десятичных знаков;
- `acDimPrecisionSeven` (7) — 7 десятичных знаков;
- `acDimPrecisionEight` (8) — 8 десятичных знаков.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRotated 0bae7d54>
(vla-get-PrimaryUnitsPrecision obj)
;;; 2
(vla-put-PrimaryUnitsPrecision obj acDimPrecisionOne)
```

8.2.452. Свойство *PrincipalDirections*

Объекты: **3DSolid**, **Region**.

Описание: главные направления. Только чтение. Возможные значения: вариант (безопасный массив), содержащий координаты трех трехмерных (двух двумерных — для области) единичных векторов.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadRegion 0bae6124>
(setq temp (vla-get-PrincipalDirections obj))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (0.427025 0.90424 -0.90424 0.427025)
```

8.2.453. Свойство *PrincipalMoments*

Объекты: **3DSolid**, **Region**.

Описание: главные моменты. Только чтение. Возможные значения: вариант (безопасный массив), содержащий моменты по трем (двум — для области) координатным осям.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadRegion 0bae6124>
(setq temp (vla-get-PrincipalMoments obj))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (1.72791e+006 2.60477e+006)
```

8.2.454. Свойство *PrinterConfigPath*

Объект: **PreferencesFiles**.

Описание: пути поиска конфигураций принтеров. Возможные значения: строка. Пути должны существовать.

```
(setq obj (vla-get-Files
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesFiles2 0bae6f10>
(vla-get-PrinterConfigPath obj)
;;; "C:\\Documents and ... enu\\Plotters\\"
(vla-put-PrinterConfigPath obj "D:\\Temp\\")
```

8.2.455. Свойство *PrinterDescPath*

Объект: **PreferencesFiles**.

Описание: пути поиска описаний принтеров. Возможные значения: строка. Пути должны существовать.

```
(setq obj (vla-get-Files
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesFiles2 0bae6f10>
(vla-get-PrinterDescPath obj)
;;; "C:\\Documents and ... enu\\Plotters\\PMP Files"
(vla-put-PrinterDescPath obj "D:\\Temp\\")
```

8.2.456. Свойство *PrinterPaperSizeAlert*

Объект: **PreferencesOutput**.

Описание: вывод предупреждения о несоответствии размера бумаги.

Возможные значения:

- `:vlax-true` — выводить;
- `:vlax-false` — не выводить.

```
(setq obj (vla-get-Output
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesOutput2 0bae1928>
(vla-get-PrinterPaperSizeAlert obj)
;;; :vlax-false
(vla-put-PrinterPaperSizeAlert obj :vlax-true)
```

8.2.457. Свойство *PrinterSpoolAlert*

Объект: **`PreferencesOutput`**.

Описание: режим предупреждений о перенаправлении печати на системный принтер в связи с ошибками порта. Возможные значения:

- `acPrinterAlwaysAlert` (0) — всегда предупреждать и писать в протокол;
- `acPrinterAlertOnce` (1) — предупредить о первой ошибке, в протокол писать все;
- `acPrinterNeverAlertLogOnce` (2) — не предупреждать, первую записать в протокол;
- `acPrinterNeverAlert` (3) — не предупреждать, не писать в протокол.

```
(setq obj (vla-get-Output
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesOutput2 0bae1138>
(vla-get-PrinterSpoolAlert obj)
;;; 0
(vla-put-PrinterSpoolAlert obj acPrinterAlertOnce)
```

8.2.458. Свойство *PrinterStyleSheetPath*

Объект: **`PreferencesFiles`**.

Описание: пути поиска файлов параметров листа. Возможные значения: строка. Пути должны существовать.

```
(setq obj (vla-get-Files
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesFiles2 0bae70f0>
(vla-get-PrinterStyleSheetPath obj)
;;; "C:\\Documents and ... enu\\Plot Styles\\"
(vla-put-PrinterStyleSheetPath obj "D:\\Temp\\")
```

8.2.459. Свойство *PrintFile*

Объект: **PreferencesFiles**.

Описание: альтернативное имя файла вывода при печати в файл с помощью пакетов старых версий. Возможные значения: строка.

```
(setq obj (vla-get-Files
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesFiles2 0bae70f0>
(vla-get-PrintFile obj)
;;; "."
(vla-put-PrintFile obj "MyPlot.plt")
```

8.2.460. Свойство *PrintSpoolerPath*

Объект: **PreferencesFiles**.

Описание: папка буфера печати. Возможные значения: строка. Путь должен существовать.

```
(setq obj (vla-get-Files
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesFiles2 0bae70f0>
(vla-get-PrintSpoolerPath obj)
;;; "C:\\Temp\\"
(vla-put-PrintSpoolerPath obj "D:\\Temp\\")
```

8.2.461. Свойство *PrintSpoolExecutable*

Объект: **PreferencesFiles**.

Описание: приложение, используемое для фоновой печати. Возможные значения: строка. Не проверяется.

```
(setq obj (vla-get-Files
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesFiles2 0bae70f0>
(vla-get-PrintSpoolExecutable obj)
;;; ""
(vla-put-PrintSpoolExecutable obj "Any string")
```

8.2.462. Свойство *ProductOfInertia*

Объекты: **3DSolid**, **Region**.

Описание: центробежный момент инерции в текущей системе координат. Только чтение. Возможные значения: вариант (безопасный массив), содержащий момен-

ты по трем координатным осям (для области — положительное вещественное число).

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcad3DSolid 0bb10874>
(setq temp (vla-get-ProductOfInertia obj))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (3.15752e+012 7.62563e+010 4.75109e+011)
```

8.2.463. Свойство *Profiles*

Объект: **Preferences**.

Описание: указатель на объект, содержащий семейство пользовательских профилей. Только чтение. Возможные значения: VLA-объект.

```
(setq obj (vla-get-Preferences (vlax-get-acad-object)))
;;; #<VLA-OBJECT IAcadPreferences 0bb129cc>
(vla-get-Profiles obj)
;;; #<VLA-OBJECT IAcadPreferencesProfiles 0bb129bc>
```

8.2.464. Свойство *PromptString*

Объект: **Attribute**.

Описание: строка подсказки. Возможные значения: строка.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadAttribute 0b038104>
(vla-get-PromptString obj)
;;; "Подсказка"
(vla-put-PromptString obj "Любая строка")
```

8.2.465. Свойство *PropertyName* 2006

Объект: **DynamicBlockReferenceProperty**.

Описание: имя свойства динамического блока. Только чтение. Возможные значения: строка.

```
(setq obj (vlax-safearray-get-element (vlax-variant-value
    (vla-GetDynamicBlockProperties (vlax-ename->vla-object (car (entsel))))) 0))
;;; #<VLA-OBJECT IAcadDynamicBlockReferenceProperty 02809484>
(vla-get-PropertyName obj)
;;; "Distance"
```

8.2.466. Свойство *ProviderName*

Объект: **SecurityParams**.

Описание: имя провайдера шифрования. Возможные значения: строка, в том числе пустая. Не проверяется.

```
(setq obj (vla-GetInterfaceObject
            (vlax-get-acad-object) "AutoCAD.SecurityParams.16"))
;;; #<VLA-OBJECT IAcadSecurityParams 0b039514>
(vla-get-ProviderName obj)
;;; ""
(vla-put-ProviderName obj "Any string")
```

8.2.467. Свойство *ProviderType*

Объект: **SecurityParams**.

Описание: имя провайдера шифрования. Возможные значения: целое число. Не проверяется.

```
(setq obj (vla-GetInterfaceObject
            (vlax-get-acad-object) "AutoCAD.SecurityParams.16"))
;;; #<VLA-OBJECT IAcadSecurityParams 0b039514>
(vla-get-ProviderType obj)
;;; 0
(vla-put-ProviderType obj 1234567890)
```

8.2.468. Свойство *ProxyImage*

Объект: **PreferencesOpenSave**.

Описание: отображение прокси-объектов (объектов других приложений). Возможные значения:

- `acProxyNotShow` (0) — выключено;
- `acProxyShow` (1) — включено;
- `acProxyBoundingBox` (2) — только габаритный прямоугольник.

```
(setq obj (vla-get-OpenSave
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesOpenSave 0b039364>
(vla-get-ProxyImage obj)
;;; 1
(vla-put-ProxyImage obj acProxyBoundingBox)
```

8.2.469. Свойство *QNewTemplateFile*

2005

Объект: **PreferencesFiles**.

Описание: шаблон, используемый по умолчанию командой QNEW (БНОВЫЙ). Возможные значения: строка (имя файла шаблона). Не проверяется.

```
(setq obj (vla-get-Files
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesFiles2 0b039060>
(vla-get-QNewTemplateFile obj)
;;; ""
(vla-put-QNewTemplateFile obj "any string")
```

8.2.470. Свойство *QuietErrorMode*

Объект: **Plot**.

Описание: вывод сообщений об ошибках печати не на экран, а в файл. Возможные значения:

- `:vlax-true` — включен;
- `:vlax-false` — выключен.

```
(setq obj (vla-get-Plot actdoc))
;;; #<VLA-OBJECT IAcadPlot 0b03902c>
(vla-get-QuietErrorMode obj)
;;; :vlax-false
(vla-put-QuietErrorMode obj :vlax-true)
```

8.2.471. Свойство *RadiiOfGyration*

Объекты: **3DSolid**, **Region**.

Описание: радиус инерции. Только чтение. Возможные значения: вариант (безопасный массив), содержащий значения по трем (двум — для области) осям.

Примечание

Получение радиусов инерции области возможно, только если она лежит в плоскости *XY* текущей системы координат.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcad3DSolid 0b038a34>
(setq temp (vla-get-RadiiOfGyration obj))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (411.092 2509.47 2541.0)
```

8.2.472. Свойство *Radius*

Объекты: **Arc**, **Circle**.

Описание: радиус. Возможные значения: положительное вещественное число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadArc 0b04ec14>
(vla-get-Radius obj)
;;; 92.4621
(vla-put-Radius obj 40)
```

8.2.473. Свойство *RadiusRatio*

Объект: **Ellipse**.

Описание: отношение полуосей. Возможные значения: от 0.0000001 до 1.0.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadEllipse 0b037bf4>
(vla-get-RadiusRatio obj)
;;; 0.790083
(vla-put-RadiusRatio obj 0.4)
```

8.2.474. Свойство *ReadOnly*

Объекты: **Document**, **DynamicBlockReferenceProperty**.

Описание: признак невозможности перезаписи документа. Только чтение. Возможные значения:

- `:vlax-true` — только чтение;
- `:vlax-false` — перезапись возможна.

```
(vla-get-ReadOnly actdoc)
;;; :vlax-false
```

8.2.475. Свойство *Red*

Объект: **AcCmColor**.

Описание: значение красной цветовой компоненты. Только чтение. Возможные значения: целое число в диапазоне 0—255.

```
(setq obj (vla-GetInterfaceObject
            (vlax-get-acad-object) "AutoCAD.AcCmColor.16"))
;;; #<VLA-OBJECT IAcadAcCmColor 0b7288a0>
(vla-get-Red obj)
;;; 0
```

8.2.476. Свойство *ReferenceCount*

Объект: **FileDependency**.

Описание: индекс внешнего файла в списке зависимостей. Только чтение. Возможные значения: положительное целое число.

```
(setq obj (vla-Item (vla-get-FileDependencies actdoc) 1))
;;; #<VLA-OBJECT IAcadFileDependency 0b728154>
(vla-get-ReferenceCount obj)
;;; 1
```

8.2.477. Свойство *RegenerateTableSuppressed* 2006

Объект: **Table**.

Описание: признак подавления регенерации блоков в таблицах. Возможные значения:

- `:vlax-true` — да;
- `:vlax-false` — нет.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable2 0280ebd4>
(vla-get-RegenerateTableSuppressed obj)
;;; :vlax-false
(vla-put-RegenerateTableSuppressed obj :vlax-true)
```

8.2.478. Свойство *RegisteredApplications*

Объект: **Document**.

Описание: указатель на семейство зарегистрированных приложений. Только чтение. Возможные значения: VLA-объект.

```
(vla-get-RegisteredApplications actdoc)
;;; #<VLA-OBJECT IAcadRegisteredApplications 0b728834>
```

8.2.479. Свойство *RenderSmoothness*

Объект: **DatabasePreferences**.

Описание: показатель качества сглаживания (относительной аппроксимации) кривых при операциях тонирования и скрытия невидимых линий. Соответствует системной переменной FACETRES. Возможные значения: число с плавающей точкой от 0.01 до 10.

```
(setq obj (vla-get-Preferences actdoc))
;;; #<VLA-OBJECT IAcadDatabasePreferences 0b729bb4>
(vla-get-RenderSmoothness obj)
;;; 0.5
(vla-put-RenderSmoothness obj 8)
```

8.2.480. Свойство *RevisionNumber*

Объект: **SummaryInfo**.

Описание: "номер редакции" документа. Возможные значения: строка, в том числе пустая.

Примечание

Свойство не обладает никакими "номерными" отличиями от обычной строки.

```
(setq obj (vla-get-SummaryInfo actdoc))
;;; #<VLA-OBJECT IAcadSummaryInfo 0b729b54>
(vla-get-RevisionNumber obj)
;;; ""
(vla-put-RevisionNumber obj "Любая строка")
```

8.2.481. Свойство *Rotation*

Объекты: **Attribute**, **AttributeReference**, **BlockRef**, **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**, **ExternalReference**, **LabelTemplate**, **MInsertBlock**, **MText**, **Raster**, **Shape**, **Text**.

Описание: угол поворота в радианах. Возможные значения: от нуля до 2π.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadText2 0b72baa4>
(vla-get-Rotation obj)
;;; 0.258362
(vla-put-Rotation obj (/ pi 5))
```

8.2.482. Свойство *RoundDistance*

Объекты: **DimAligned**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**.

Описание: точность округления основных единиц. Переопределяет значение, установленное системной переменной DIMRND. Возможные значения: неотрицательное вещественное число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRotated 0b73f9c4>
(vla-get-RoundDistance obj)
;;; 0.0
(vla-put-RoundDistance obj 2.0)
```

8.2.483. Свойство *RowHeight*

2005

Объект: **Table**.

Описание: высота всех строк. Только установка. Возможные значения: положительное вещественное число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0b766e84>
(vla-put-RowHeight obj 20)
```

8.2.484. Свойство *Rows*

Объекты: **MInsertBlock**, **Table**.

Описание: количество строк (установка только для `MInsertBlock`). Возможные значения: положительное целое число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadMInsertBlock 0b75c304>
(vla-get-Rows obj)
;;; 4
(vla-put-Rows obj 5)
```

8.2.485. Свойство *RowSpacing*

Объект: **MInsertBlock**.

Описание: расстояние между строками. Возможные значения: вещественное число, включая 0 и отрицательные.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadMInsertBlock 0b75c304>
(vla-get-RowSpacing obj)
;;; 99.6205
(vla-put-RowSpacing obj 150)
```

8.2.486. Свойство *SaveAsType*

Объект: **PreferencesOpenSave**.

Описание: формат сохранения документа.

Возможные значения:

- acUnknown (–1) — неизвестная версия или файл поврежден (только чтение);
- acR12_DXF (1) — R12 DXF;
- ac2000_dwg (12) — R15 DWG;
- acR15_dwg (12) — R15 DWG;
- ac2000_dxf (13) — R15 DXF;
- acR15_dxf (13) — R15 DXF;
- ac2000_Template (14) — R15 DWT;
- acR15_Template (14) — R15 DWT;
- acNative (24) — "родной" формат;
- ac2004_dwg (24) — R16 DWG;
- acR18_dwg (24) — R16 DWG;
- ac2004_dxf (25) — R16 DXF;
- acR18_dxf (25) — R16 DXF;
- ac2004_Template (26) — R16 DWT;
- acR18_Template (26) — R16 DWT.

Примечание

Следующие значения отменены, как устаревшие: acR13_DWG (4), acR13_DXF (5), acR14_DWG (8) и acR14_DXF (9). Обозначение R15 соответствует коммерческим названиям AutoCAD 2000, 2000i, 2002 и всем вертикальным решениям, базирующимся на этой версии, R16 — коммерческим названиям AutoCAD 2004, 2005, 2006 и надстройкам над ними.

```
(setq obj (vla-get-OpenSave
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesOpenSave 0b524ab4>
(vla-get-SaveAsType obj)
;;; 24
(vla-put-SaveAsType obj ac2000_dxf)
```

8.2.487. Свойство *Saved*

Объект: **Document**.

Описание: признак отсутствия несохраненных изменений. Только чтение. Возможные значения:

- :vlax-true — да;
- :vlax-false — нет.

```
(vla-get-Saved actdoc)
;;; :vlax-true
```

8.2.488. Свойство *SavePreviewThumbnail*

Объект: **PreferencesOpenSave**.

Описание: сохранение растрового образца вместе с документом. Соответствует системной переменной RASTERPREVIEW. Возможные значения:

- `:vlax-true` — да;
- `:vlax-false` — нет.

```
(setq obj (vla-get-OpenSave
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesOpenSave 0b524df4>
(vla-get-SavePreviewThumbnail obj)
;;; :vlax-true
    (vla-put-SavePreviewThumbnail obj :vlax-false)
```

8.2.489. Свойство *ScaleFactor*

Объекты: **Attribute**, **AttributeReference**, **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**, **Leader**, **Raster**, **Shape**, **Text**, **Tolerance**.

Описание: масштабный коэффициент (для **Attribute**, **AttributeReference**, **Text** — по горизонтали). Для размерных примитивов переопределяет значение, установленное системной переменной DIMSCALE. Возможные значения: положительное вещественное число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRotated 0b523e54>
(vla-get-ScaleFactor obj)
;;; 1.0
(vla-put-ScaleFactor obj 1.5)
```

8.2.490. Свойство *ScaleHeight*

Объект: **Ole**.

Описание: относительный вертикальный масштаб вставки. Возможные значения: положительное вещественное число (0.0 — только чтение).

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadOle 0b9b01a4>
(vla-get-ScaleHeight obj)
;;; 0.0
(vla-put-ScaleHeight obj 150)
```

8.2.491. Свойство *ScaleLineweights*

Объекты: **Layout**, **PlotConfiguration**.

Описание: масштабирование весов линий при печати. Возможные значения:

- :vlax-true — да;
- :vlax-false — нет.

```
(setq obj (vla-get-ActiveLayout actdoc))
;;; #<VLA-OBJECT IAcadLayout 0b9bd854>
(vla-get-ScaleLineweights obj)
;;; :vlax-false
(vla-put-ScaleLineweights obj :vlax-true)
```

8.2.492. Свойство *ScaleWidth*

2005

Объект: **Ole**.

Описание: относительный горизонтальный масштаб вставки. Возможные значения: положительное вещественное число (0.0 — только чтение).

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadOle 0b9b01a4>
(vla-get-ScaleWidth obj)
;;; 6666.67
(vla-put-ScaleWidth obj 8000)
```

8.2.493. Свойство *Schema*

Объекты: **LabelTemplate**, **LinkTemplate**, **Query**.

Описание: имя схемы. Возможные значения: строка.

Примечание

Свойство доступно для изменения только при отсутствии связей, использующих шаблон, с которым производятся манипуляции.

```
(setq DBConn (vla-GetInterfaceObject (vlax-get-acad-object)
               "CAO.DbConnect.16"))
;;; #<VLA-OBJECT IDbConnect 0573c770>
(setq DBLinksT (vlax-invoke DBConn 'GetLinkTemplates))
;;; #<VLA-OBJECT ILinkTemplates 05739c90>
(setq DBLinkT (vla-Item DBLinksT "Link3")) ;_был создан заранее
;;; #<VLA-OBJECT ILinkTemplate 05740260>
(vlax-get-property DBLinkT 'Schema)
;;; ""
(vlax-put-property DBLinkT 'Schema "Any string")
```

8.2.494. Свойство *SCMCommandMode*

Объект: **PreferencesUser**.

Описание: смысловая нагрузка правой клавиши в режиме выполнения команды. Соответствует системной переменной SHORTCUTMENU. Возможные значения:

- acEnter (0) — аналог клавиши <Enter>;
- acEnableSCMOptions (1) — показ контекстного меню при наличии опций;
- acEnableSCM (2) — показ контекстного меню всегда.

```
(setq obj (vla-get-User
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesUser 0b9b3af8>
(vla-get-SCMCommandMode obj)
;;; 1
(vla-put-SCMCommandMode obj acEnableSCM)
```

8.2.495. Свойство *SCMDefaultMode*

Объект: **PreferencesUser**.

Описание: смысловая нагрузка правой клавиши при отсутствии выбранных примитивов и вне контекста выполнения какой-либо команды. Соответствует системной переменной SHORTCUTMENU. Возможные значения:

- acRepeatLastCommand (0) — повтор последней команды;
- acSCM (1) — показ контекстного меню.

```
(setq obj (vla-get-User
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesUser 0b9b3af8>
(vla-get-SCMDefaultMode obj)
;;; 1
(vla-put-SCMDefaultMode obj acRepeatLastCommand)
```

8.2.496. Свойство *SCMEditMode*

Объект: **PreferencesUser**.

Описание: смысловая нагрузка правой клавиши при наличии выбранных примитивов, но вне контекста выполнения какой-либо команды. Соответствует системной переменной SHORTCUTMENU. Возможные значения:

- acEdRepeatLastCommand (0) — повтор последней команды;
- acEdSCM (1) — показ контекстного меню.

```
(setq obj (vla-get-User
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesUser 0b9b3af8>
(vla-get-SCMEditMode obj)
;;; 1
(vla-put-SCMEditMode obj acEdRepeatLastCommand)
```

8.2.497. Свойство *SCMTimeMode*

Объект: **PreferencesUser**.

Описание: чувствительность правой клавиши к длительности нажатия. Соответствует системной переменной SHORTCUTMENU. Возможные значения:

- `:vlax-true` — да;
- `:vlax-false` — нет.

```
(setq obj (vla-get-User
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesUser 0b9b3af8>
(vla-get-SCMTimeMode obj)
;;; :vlax-false
(vla-put-SCMTimeMode obj :vlax-true)
```

8.2.498. Свойство *SCMTimeValue*

Объект: **PreferencesUser**.

Описание: задержка удержания правой клавиши в миллисекундах, которая распознается как длительная. Возможные значения: целое число от 100 до 10 000.

```
(setq obj (vla-get-User
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesUser 0b9b3af8>
(vla-get-SCMTimeValue obj)
;;; 250
(vla-put-SCMTimeValue obj 400)
```

8.2.499. Свойство *SecondPoint*

Объекты: **Ray**, **XLine**.

Описание: координаты второй точки. Возможные значения: вариант (безопасный массив), содержащий три координаты.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadXline 0be3dbb4>
```

```
(setq temp (vla-get-SecondPoint obj))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (2646.67 794.741 0.0)
(vla-put-SecondPoint obj (vlax-3d-point (getpoint)))
```

8.2.500. Свойство *SegmentPerPolyline*

Объект: **DatabasePreferences**.

Описание: число прямолинейных сегментов, которыми аппроксимируется полилиния при сглаживании сплайном в команде PEDIT (ПОЛРЕД). Соответствует положительным значениям системной переменной SPLINESEGS. Возможные значения: целое число от 1 до 32 767.

```
(setq obj (vla-get-Preferences actdoc))
;;; #<VLA-OBJECT IAcadDatabasePreferences 0be3b3b4>
(vla-get-SegmentPerPolyline obj)
;;; 8
(vla-put-SegmentPerPolyline obj 12)
```

8.2.501. Свойство *Selection*

Объект: **Preferences**.

Описание: указатель на объект, содержащий настройки режимов выбора. Только чтение. Возможные значения: VLA-объект.

```
(setq obj (vla-get-Preferences (vlax-get-acad-object)))
;;; #<VLA-OBJECT IAcadPreferences 0be5481c>
(vla-get-Selection obj)
;;; #<VLA-OBJECT IAcadPreferencesSelection 0be54810>
```

8.2.502. Свойство *SelectionSets*

Объект: **Document**.

Описание: указатель на семейство наборов. Только чтение. Возможные значения: VLA-объект.

```
(vla-get-SelectionSets actdoc)
;;; #<VLA-OBJECT IAcadSelectionSets 0be546e4>
```

8.2.503. Свойство *SerialNumber*

Объект: **SecurityParams**.

Описание: серийный номер цифрового сертификата. Возможные значения: строка, в том числе пустая.

```
(setq obj (vla-GetInterfaceObject
            (vlax-get-acad-object) "AutoCAD.SecurityParams.16"))
;;; #<VLA-OBJECT IAcadSecurityParams 0be57894>
(vla-get-SerialNumber obj)
;;; ""
(vla-put-SerialNumber obj "Any string")
```

8.2.504. Свойство *ShadePlot*

Объект: **PViewport**.

Описание: раскрашивание при печати. Возможные значения:

- acShadePlotAsDisplayed (0) — как отображается;
- acShadePlotWireframe (1) — каркасное представление;
- acShadePlotHidden (2) — со скрытием невидимых линий;
- acShadePlotRendered (3) — тонированное.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadPViewport2 0a8a1414>
(vla-get-ShadePlot obj)
;;; 0
(vla-put-ShadePlot obj acShadePlotHidden)
```

8.2.505. Свойство *SheetView*

2005

Объект: **PViewport**.

Описание: связь с видом в подшивке. Возможные значения: VLA-объект вида.

Примечание

Свойство не документировано. Для установки требуются специальные условия.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadPViewport2 0ba129f4>
(vla-get-SheetView obj)
;;; #<VLA-OBJECT IAcadView2 0ba15ed4>
(vla-put-SheetView obj (vla-Item (vla-get-Views actdoc) 2))
```

8.2.506. Свойство *ShortcutMenu*

Объект: **PopupMenu**.

Описание: признак контекстного меню. Только чтение. Возможные значения:

- :vlax-true — контекстное;
- :vlax-false — падающее.

```
(setq obj (vla-Item (vla-get-Menus (vla-Item (vla-get-MenuGroups
    (vlax-get-acad-object)) "acad")) 9))
;;; #<VLA-OBJECT IAcadPopupMenu 010aeb18>
(vla-get-ShortcutMenu obj)
;;; :vlax-false
```

8.2.507. Свойство *ShortCutMenuDisplay*

Объект: **PreferencesUser**.

Описание: показ контекстного меню при щелчке правой кнопкой мыши в графическом экране. Соответствует системной переменной SHORTCUTMENU. Возможные значения:

- `:vlax-true` — показывать;
- `:vlax-false` — не показывать.

```
(setq obj (vla-get-User
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesUser 0b2e4198>
(vla-get-ShortCutMenuDisplay obj)
;;; :vlax-true
(vla-put-ShortCutMenuDisplay obj :vlax-false)
```

8.2.508. Свойство *Show*

Объект: **DynamicBlockReferenceProperty**.

Описание: признак отображения свойства. Только чтение. Возможные значения:

- `:vlax-true` — да;
- `:vlax-false` — нет.

```
(setq obj (vlax-safearray-get-element (vlax-variant-value
    (vla-GetDynamicBlockProperties (vlax-ename->vla-object (car (entsel))))) 0))
;;; #<VLA-OBJECT IAcadDynamicBlockReferenceProperty 02809484>
(vla-get-Show obj)
;;; :vlax-true
```

8.2.509. Свойство *ShowPlotStyles*

Объекты: **Layout**, **PlotConfiguration**.

Описание: отображение с применением стилей печати. Возможные значения:

- `:vlax-true` — включено;
- `:vlax-false` — выключено.

```
(setq obj (vla-get-ActiveLayout actdoc))
;;; #<VLA-OBJECT IAcadLayout 0b2ebdc4>
(vla-get-ShowPlotStyles obj)
;;; :vlax-false
(vla-put-ShowPlotStyles obj :vlax-true)
(vla-Regen actdoc acAllViewports)
```

8.2.510. Свойство *ShowProxyDialogBox*

Объект: **PreferencesOpenSave**.

Описание: вывод сообщения о прокси-объектах при открытии рисунка с объектами других приложений. Соответствует системной переменной PROXYNOTICE. Возможные значения:

- `:vlax-true` — включен;
- `:vlax-false` — выключен.

```
(setq obj (vla-get-OpenSave
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesOpenSave 0b2edc74>
(vla-get-ShowProxyDialogBox obj)
;;; :vlax-true
(vla-put-ShowProxyDialogBox obj :vlax-false)
```

8.2.511. Свойство *ShowRasterImage*

Объект: **PreferencesDisplay**.

Описание: отображение растровых изображений в процессе панорамирования и зумирования. Соответствует системной переменной RTDISPLAY. Возможные значения:

- `:vlax-true` — полностью;
- `:vlax-false` — только рамка.

```
(setq obj (vla-get-Display
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesDisplay 0b2edc1c>
(vla-get-ShowRasterImage obj)
;;; :vlax-false
(vla-put-ShowRasterImage obj :vlax-true)
```

8.2.512. Свойство *ShowRotation*

Объект: **Raster**.

Описание: отображение растрового изображения при ненулевом повороте.

Возможные значения:

- `:vlax-true` — включено;
- `:vlax-false` — выключено.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadRasterImage 0b2f9d14>
(vla-get-ShowRotation obj)
;;; :vlax-true
(vla-put-ShowRotation obj :vlax-false)
```

8.2.513. Свойство *ShowWarningMessages*

Объект: **PreferencesSystem**.

Описание: восстановление вывода предупреждающего сообщения **Don't Display This Warning Again** (Не выводить это предупреждение в дальнейшем). Возможные значения:

- `:vlax-true` — восстановить вывод;
- `:vlax-false` — отказаться от вывода.

```
(setq obj (vla-get-System
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesSystem 0b309cb4>
(vla-get-ShowWarningMessages obj)
;;; :vlax-true
(vla-put-ShowWarningMessages obj :vlax-false)
```

8.2.514. Свойство *SingleDocumentMode*

Объект: **PreferencesSystem**.

Описание: признак однодокументного режима. Соответствует системной переменной SDI. Возможные значения:

- `:vlax-true` — однодокументный режим;
- `:vlax-false` — многодокументный режим.

Примечание

Установка однодокументного режима не допускается (приводит к ошибке), если уже открыто более одного файла.

```
(setq obj (vla-get-System
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesSystem 0b309cb4>
```

```
(vla-get-SingleDocumentMode obj)
;;; :vlax-false
(vla-put-SingleDocumentMode obj :vlax-true)
```

8.2.515. Свойство *SnapBasePoint*

Объекты: **PViewport**, **Viewport**.

Описание: точка начала сетки для режимов **SNAP** (ШАГ) и **GRID** (СЕТКА). Соответствует системной переменной SNAPBASE. Возможные значения: вариант (безопасный массив), содержащий координаты двумерной точки.

```
(setq obj (vla-get-ActiveViewport actdoc))
;;; #<VLA-OBJECT IAcadViewport 0b3c02a4>
(setq temp (vla-get-SnapBasePoint obj))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (0.0 0.0)
(vla-put-SnapBasePoint obj (vlax-safearray-fill
    (vlax-make-safearray vlax-vbDouble '(0 . 1)) '(150 120)))
(vla-Regen actdoc acAllViewports)
```

8.2.516. Свойство *SnapOn*

Объекты: **PViewport**, **Viewport**.

Описание: состояние режима **SNAP** (ШАГ). Соответствует системной переменной SNAPMODE. Возможные значения:

- `:vlax-true` — включен;
- `:vlax-false` — выключен.

```
(setq obj (vla-get-ActivePViewport actdoc))
;;; #<VLA-OBJECT IAcadPViewport2 0ba4f5f4>
(vla-get-SnapOn obj)
;;; :vlax-false
(vla-put-SnapOn obj :vlax-true)
```

8.2.517. Свойство *SnapRotationAngle*

Объекты: **PViewport**, **Viewport**.

Описание: угол наклона сетки в радианах. Соответствует системной переменной SNAPANG. Возможные значения: от 0 до 2π.

```
(setq obj (vla-get-ActivePViewport actdoc))
;;; #<VLA-OBJECT IAcadPViewport2 0ba4f5f4>
```

```
(vla-get-SnapRotationAngle obj)
;;; 0.0
(vla-put-SnapRotationAngle obj (/ pi 3))
```

8.2.518. Свойство *SolidFill*

Объект: **DatabasePreferences**.

Описание: режим заливки для мультилиний, полос, фигур, полилиний и штриховок типа Solid. Соответствует системной переменной FILLMODE. Возможные значения:

- `:vlax-true` — включен;
- `:vlax-false` — выключен.

```
(setq obj (vla-get-Preferences actdoc))
;;; #<VLA-OBJECT IAcadDatabasePreferences 0bb32114>
(vla-get-SolidFill obj)
;;; :vlax-true
(vla-put-SolidFill obj :vlax-false)
(vla-Regen actdoc acAllViewports)
```

8.2.519. Свойство *StandardScale*

Объекты: **Layout**, **PViewport**, **PlotConfiguration**.

Описание: масштаб из списка стандартных значений. Возможные значения:

для `PViewport`

- `acVpScaleToFit` (0) — вписанный;
- `acVpCustomScale` (1) — пользовательский;
- `acVp1_1` (2) — 1:1;
- `acVp1_2` (3) — 1:2;
- `acVp1_4` (4) — 1:4;
- `acVp1_8` (5) — 1:8;
- `acVp1_10` (6) — 1:10;
- `acVp1_16` (7) — 1:16;
- `acVp1_20` (8) — 1:20;
- `acVp1_30` (9) — 1:30;
- `acVp1_40` (10) — 1:40;
- `acVp1_50` (11) — 1:50;
- `acVp1_100` (12) — 1:100;

для `Layout` и `PlotConfiguration`

- `acScaleToFit` (0) — вписанный;
- `ac1_128in_1ft` (1) — 1/128"=1';
- `ac1_64in_1ft` (2) — 1/64"=1';
- `ac1_32in_1ft` (3) — 1/32"=1';
- `ac1_16in_1ft` (4) — 1/16"=1';
- `ac3_32in_1ft` (5) — 3/32"=1';
- `ac1_8in_1ft` (6) — 1/8"=1';
- `ac3_16in_1ft` (7) — 3/16"=1';
- `ac1_4in_1ft` (8) — 1/4"=1';
- `ac3_8in_1ft` (9) — 3/8"=1';
- `ac1_2in_1ft` (10) — 1/2"=1';
- `ac3_4in_1ft` (11) — 3/4"=1';
- `ac1in_1ft` (12) — 1"=1';

- acVp2_1 (13) — 2:1;
- acVp4_1 (14) — 4:1;
- acVp8_1 (15) — 8:1;
- acVp10_1 (16) — 10:1;
- acVp100_1 (17) — 100:1;
- acVp1_128in_1ft (18) — 1/128"=1';
- acVp1_64in_1ft (19) — 1/64"=1';
- acVp1_32in_1ft (20) — 1/32"=1';
- acVp1_16in_1ft (21) — 1/16"=1';
- acVp3_32in_1ft (22) — 3/32"=1';
- acVp1_8in_1ft (23) — 1/8"=1';
- acVp3_16in_1ft (24) — 3/16"=1';
- acVp1_4in_1ft (25) — 1/4"=1';
- acVp3_8in_1ft (26) — 3/8"=1';
- acVp1_2in_1ft (27) — 1/2"=1';
- acVp3_4in_1ft (28) — 3/4"=1';
- acVp1in_1ft (29) — 1"=1';
- acVp3in_1ft (31) — 3"=1';
- acVp6in_1ft (32) — 6"=1';
- acVp1ft_1ft (33) — 1'=1'.
- ac3in_1ft (13) — 3"=1';
- ac6in_1ft (14) — 6"=1';
- ac1ft_1ft (15) — 1'=1';
- ac1_1 (16) — 1:1;
- ac1_2 (17) — 1:2;
- ac1_4 (18) — 1:4;
- ac1_8 (19) — 1:8;
- ac1_10 (20) — 1:10;
- ac1_16 (21) — 1:16;
- ac1_20 (22) — 1:20;
- ac1_30 (23) — 1:30;
- ac1_40 (24) — 1:40;
- ac1_50 (25) — 1:50;
- ac1_100 (26) — 1:100;
- ac2_1 (27) — 2:1;
- ac4_1 (28) — 4:1;
- ac8_1 (29) — 8:1;
- ac10_1 (30) — 10:1;
- ac100_1 (31) — 100:1;

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadPViewport2 0bb3ad94>
(vla-get-StandardScale obj)
;;; 1
(vla-put-StandardScale obj ac1_20)
```

8.2.520. Свойство *StartAngle*

Объекты: **Arc**, **Ellipse**.

Описание: начальный угол. Возможные значения: в диапазоне ±1.99999998408450585519π (для полного эллипса — 0).

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadArc 0b500b04>
(vla-get-StartAngle obj)
;;; 5.0369
(vla-put-StartAngle obj (* pi 1.5))
```

8.2.521. Свойство *StartParameter*

Объект: **Ellipse**.

Описание: начальный параметр. Возможные значения: в диапазоне ±1.99999996816901137739π (для полного эллипса — 0).

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadEllipse 0bb20bd4>
(vla-get-StartParameter obj)
;;; 0.0
(vla-put-StartParameter obj 4)
```

8.2.522. Свойство *StartPoint*

Объекты: **Arc**, **Ellipse**, **Line**.

Описание: координаты начальной точки. Возможные значения: вариант (безопасный массив) с координатами трехмерной точки.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadLine 0bb5dc54>
(setq temp (vla-get-StartPoint obj))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (4175.8 1210.48 0.0)
(vla-put-StartPoint obj (vlax-3d-point (getpoint)))
```

8.2.523. Свойство *StartTangent*

Объект: **Spline**.

Описание: вектор касательной в начальной точке сплайна. Возможные значения: вариант (безопасный массив) с координатами трехмерной точки.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadSpline 0b515084>
(setq temp (vla-get-StartTangent obj))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (-0.82927 0.558848 0.0)
(vla-put-StartTangent obj (vlax-3d-point '(1.0 1.0 0.0)))
```

8.2.524. Свойство *StatusId*

Объект: **Application**.

Описание: активность дополнительно указываемого видового экрана. Только чтение.

Возможные значения:

- `:vlax-true` — активен;
- `:vlax-false` — не активен.

Примечание

Данное свойство является исключением, имеющим дополнительный аргумент при получении значения. Фактически было бы правильнее отнести его к методам, поскольку получаемое значение относится не к тому объекту, к которому происходит обращение.

```
(setq obj (vlax-get-acad-object))
;;; #<VLA-OBJECT IAcadApplication 00b9d604>
(vla-get-StatusId obj (vla-get-ActiveViewport actdoc))
;;; :vlax-false
```

8.2.525. Свойство *StoreSQLIndex*

Объект: **PreferencesSystem**.

Описание: сохранение в рисунке SQL-индекса. Возможные значения:

- `:vlax-true` — включено;
- `:vlax-false` — выключено.

```
(setq obj (vla-get-System
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesSystem 0b31cb34>
(vla-get-StoreSQLIndex obj)
;;; :vlax-true
(vla-put-StoreSQLIndex obj :vlax-false)
```

8.2.526. Свойство *StyleName*

Объекты: **Attribute**, **AttributeReference**, **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**, **Leader**, **MLine**, **MText**, **Table**, **Text**, **Tolerance**.

Описание: имя стиля текста, размеров, мультилиний, таблиц. Для мультилиний только чтение. Возможные значения: строка.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadText2 0b314404>
(vla-get-StyleName obj)
;;; "Standard"
(vla-put-StyleName obj "MyTextStyle")
```

8.2.527. Свойство *StyleSheet*

Объекты: **Layout**, **PlotConfiguration**.

Описание: имя таблицы стилей печати. Возможные значения: строка.

```
(setq obj (vla-get-ActiveLayout actdoc))
;;; #<VLA-OBJECT IAcadLayout 0b312114>
(vla-get-StyleSheet obj)
;;; "monochrome.stb"
(vla-put-StyleSheet obj "")
```

8.2.528. Свойство *Subject*

Объекты: **SecurityParams**, **SummaryInfo**.

Описание: имя субъекта цифровой подписи или тема в свойствах рисунка. Возможные значения: строка, в том числе пустая.

```
(setq obj (vla-get-SummaryInfo actdoc))
;;; #<VLA-OBJECT IAcadSummaryInfo 0b314314>
(vla-get-Subject obj)
;;; ""
(vla-put-Subject obj "Любая строка")
```

8.2.529. Свойство *SubMenu*

Объект: **PopupMenuItem**.

Описание: подменю, связанное с данным пунктом меню. Только чтение. Возможные значения: VLA-объект подменю.

```
(setq obj (vla-Item (vla-Item (vla-get-Menus
            (vla-Item (vla-get-MenuGroups (vlax-get-acad-object))
                      "acad")) 9) 3))
;;; #<VLA-OBJECT IAcadPopupMenuItem 7ef2cc68>
(vla-get-SubMenu obj)
;;; #<VLA-OBJECT IAcadPopupMenu 010ae138>
```

8.2.530. Свойство *SummaryInfo*

Объект: **Document**.

Описание: указатель на объект, содержащий сопроводительную информацию. Только чтение. Возможные значения: VLA-объект.

```
(vla-get-SummaryInfo actdoc)
;;; #<VLA-OBJECT IAcadSummaryInfo 0b314314>
```

8.2.531. Свойство *SupportPath*

Объект: **PreferencesFiles**.

Описание: пути к файлам поддержки. Возможные значения: строка. Проверяется.

```
(setq obj (vla-get-Files
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesFiles2 0b322a90>
(setq temp (vla-get-SupportPath obj))
;;; "C:\\Docum ... \\Support; ... C:\\Prog ... \\Help"
(vla-put-SupportPath obj (strcat temp ";D:\\Temp"))
```

8.2.532. Свойство *SuppressLeadingZeros*

Объекты: **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**.

Описание: подавление ведущих нулей. Переопределяет значение, установленное системной переменной DIMZIN. Возможные значения:

- `:vlax-true` — включено;
- `:vlax-false` — выключено.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRotated 0b3260d4>
(vla-get-SuppressLeadingZeros obj)
;;; :vlax-false
(vla-put-SuppressLeadingZeros obj :vlax-true)
```

8.2.533. Свойство *SuppressTrailingZeros*

Объекты: **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**.

Описание: подавление хвостовых нулей. Переопределяет значение, установленное системной переменной DIMZIN. Возможные значения:

- `:vlax-true` — включено;
- `:vlax-false` — выключено.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRotated 0b3260d4>
(vla-get-SuppressTrailingZeros obj)
;;; :vlax-true
(vla-put-SuppressTrailingZeros obj :vlax-false)
```

8.2.534. Свойство *SuppressZeroFeet*

Объекты: **DimAligned**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**.

Описание: подавление футовых нулей. Переопределяет значение, установленное системной переменной DIMZIN. Возможные значения:

- `:vlax-true` — подавлять;
- `:vlax-false` — не подавлять.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimAligned 0ae19584>
(vla-get-SuppressZeroFeet obj)
;;; :vlax-true
(vla-put-SuppressZeroFeet obj :vlax-false)
```

8.2.535. Свойство *SuppressZeroInches*

Объекты: **DimAligned**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**.

Описание: подавление дюймовых нулей. Переопределяет значение, установленное системной переменной DIMZIN. Возможные значения:

- `:vlax-true` — подавлять;
- `:vlax-false` — не подавлять.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimAligned 0ae19584>
(vla-get-SuppressZeroInches obj)
;;; :vlax-true
(vla-put-SuppressZeroInches obj :vlax-false)
```

8.2.536. Свойство *SymbolPosition*

Объект: **DimArcLength**.

Описание: положение символа размера. Возможные значения:

- `acSymInFront` (0) — спереди;
- `acSymAbove` (1) — сверху;
- `acSymNone` (2) — нет.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadMLine2 027db2b4>
(vla-get-SymbolPosition obj)
;;; 0
(vla-put-SymbolPosition obj acSymNone)
```

8.2.537. Свойство *System*

Объект: **Preferences**.

Описание: указатель на объект, содержащий системные настройки. Только чтение. Возможные значения: VLA-объект.

```
(setq obj (vla-get-Preferences (vlax-get-acad-object)))
;;; #<VLA-OBJECT IAcadPreferences 0b322dec>
(vla-get-System obj)
;;; #<VLA-OBJECT IAcadPreferencesSystem 0b322de4>
```

8.2.538. Свойство *Table*

Объекты: **LabelTemplate**, **LinkTemplate**, **Query**.

Описание: имя связанной таблицы. Возможные значения: строка.

Примечание

Свойство доступно для изменения только при отсутствии связей, использующих шаблон, с которым производятся манипуляции. При установке значение не проверяется, но для функционирования наличие источника данных обязательно.

```
(setq DBConn (vla-GetInterfaceObject (vlax-get-acad-object)
              "CAO.DbConnect.16"))
;;; #<VLA-OBJECT IDbConnect 05811f70>
(setq DBLinksT (vlax-invoke DBConn 'GetLinkTemplates))
;;; #<VLA-OBJECT ILinkTemplates 05810b90>
(setq DBLinkT (vla-Item DBLinksT "Link3")) ;_был создан заранее
;;; #<VLA-OBJECT ILinkTemplate 058108e0>
(vlax-get-property DBLinkT 'Table)
;;; "Computer"
(vlax-put-property DBLinkT 'Table "Employee")
```

8.2.539. Свойство *TablesReadOnly*

Объект: **PreferencesSystem**.

Описание: открытие таблиц базы данных "только для чтения". Возможные значения:

- `:vlax-true` — включено;
- `:vlax-false` — выключено.

```
(setq obj (vla-get-System
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesSystem 0aedca44>
```

```
(vla-get-TablesReadOnly obj)
;;; :vlax-false
(vla-put-TablesReadOnly obj :vlax-true)
```

8.2.540. Свойство *TableStyleOverrides*

2005

Объект: **Table**.

Описание: переопределение стиля. Только чтение. Возможные значения: список кодов переопределения стиля в виде варианта (безопасного массива).

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 0aedc344>
(setq temp (vla-get-TableStyleOverrides obj))
;;; #<variant 8195 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (2 4)
```

8.2.541. Свойство *TabOrder*

Объект: **Layout**.

Описание: номер вкладки в порядке расположения в списке вкладок. Возможные значения: положительное целое.

Примечание

Вкладка **Model** (Модель) всегда имеет номер 0. Для других вкладок при установке значение не может быть больше, чем общее количество вкладок листов.

```
(setq obj (vla-get-ActiveLayout actdoc))
;;; #<VLA-OBJECT IAcadLayout 123cb6a4>
(vla-get-TabOrder obj)
;;; 2
(vla-put-TabOrder obj 4)
```

8.2.542. Свойство *TagString*

Объекты: **Attribute**, **AttributeReference**, **PopupMenu**, **PopupMenuItem**, **Toolbar**, **ToolbarItem**.

Описание: строковый тег. Возможные значения: строка.

```
(setq obj (vlax-ename->vla-object (car (nentsel))))
;;; #<VLA-OBJECT IAcadAttributeReference 0aedafb4>
(vla-get-TagString obj)
;;; "TEST2"
(vla-put-TagString obj "Any string")
```

8.2.543. Свойство *Target*

Объекты: **PViewport**, **View**, **Viewport**.

Описание: точка цели. Возможные значения: вариант (безопасный массив), содержащий координаты трехмерной точки.

```
(setq obj (vla-get-ActivePViewport actdoc))
;;; #<VLA-OBJECT IAcadPViewport2 123f7974>
(setq temp (vla-get-Target obj))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (0.0 0.0 0.0)
(vla-put-Target obj (vlax-3d-point (getpoint)))
```

8.2.544. Свойство *TempFileExtension*

Объект: **PreferencesOpenSave**.

Описание: расширение имен временных файлов. Возможные значения: строка длиной от 0 до 3 символов, легальных для использования в расширении имен файлов.

```
(setq obj (vla-get-OpenSave
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesOpenSave 123eab34>
(vla-get-TempFileExtension obj)
;;; "ac$"
(vla-put-TempFileExtension obj "pl$")
```

8.2.545. Свойство *TempFilePath*

Объект: **PreferencesFiles**.

Описание: путь для временных файлов. Возможные значения: строка, полный путь.

Примечание

При установке путь должен существовать.

```
(setq obj (vla-get-Files
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesFiles2 123eaae0>
(vla-get-TempFilePath obj)
;;; "C:\\Temp\\"
(vla-put-TempFilePath obj "C:\\Temp\\DWG_TMP\\")
```

8.2.546. Свойство *TemplateDwgPath*

Объект: **PreferencesFiles**.

Описание: путь к файлам шаблонов, используемый Мастером начала работы. Возможные значения: строка, полный путь. Не проверяется.

```
(setq obj (vla-get-Files
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesFiles2 123eaae0>
(vla-get-TemplateDwgPath obj)
;;; "C:\\Docu ... 2005\\enu\\Template"
(vla-put-TemplateDwgPath obj "C:\\Templates\\")
```

8.2.547. Свойство *TempXrefPath*

Объект: **PreferencesFiles**.

Описание: путь к файлам копий (вариант загрузки внешних ссылок). Возможные значения: строка, полный путь. Не проверяется.

```
(setq obj (vla-get-Files
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesFiles2 123eaae0>
(vla-get-TempXrefPath obj)
;;; "C:\\Temp\\"
(vla-put-TempXrefPath obj "C:\\XRefs\\")
```

8.2.548. Свойство *Text*

Объект: **Query**.

Описание: текст запроса. Возможные значения: строка.

```
(setq DBConn (vla-GetInterfaceObject (vlax-get-acad-object)
               "CAO.DbConnect.16"))
;;; #<VLA-OBJECT IDbConnect 05811f70>
(setq DBQueries (vlax-invoke DBConn 'GetQueries))
;;; #<VLA-OBJECT IQueries 05860ed0>
(setq DBQuery (vla-Item DBQueries "Query1")) ;_был создан заранее
;;; #<VLA-OBJECT IQuery 05861f30>
(vlax-get-property DBQuery 'Text)
;;; "SELECT * FROM Inventory\r\n\tWHERE Manufacturer = 'Office Master'"
(vlax-put-property DBQuery 'Text
    "SELECT * FROM Inventory\r\n\tWHERE Price > 150")
```

8.2.549. Свойство *TextAlignmentPoint*

Объекты: **Attribute**, **AttributeReference**, **Text**.

Описание: вторая точка выравнивания. Для выравнивания влево всегда (0.0 0.0 0.0) и только чтение. Возможные значения: вариант (безопасный массив), содержащий координаты трехмерной точки.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadText2 123e2824>
(setq temp (vla-get-TextAlignmentPoint obj))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (2175.43 -100.0 0.0)
(vla-put-TextAlignmentPoint obj (vlax-3d-point (getpoint)))
```

8.2.550. Свойство *TextAttachment*

Объект: **LabelTemplate**.

Описание: выравнивание точки привязки текста. Возможные значения: см. свойство AttachmentPoint.

```
(setq DBConn (vla-GetInterfaceObject (vlax-get-acad-object)
              "CAO.DbConnect.16"))
;;; #<VLA-OBJECT IDbConnect 05811f70>
(setq DBLablsT (vlax-invoke DBConn 'GetLabelTemplates))
;;; #<VLA-OBJECT ILabelTemplates 058415f0>
(setq DBLablT (vla-Item DBLablsT 0)) ;_был создан заранее
;;; #<VLA-OBJECT ILabelTemplate 05819e20>
(vlax-get-property DBLablT 'TextAttachment)
;;; 1
(vlax-put-property DBLablT 'TextAttachment acAttachmentPointBottomLeft)
```

8.2.551. Свойство *TextColor*

Объекты: **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**, **Tolerance**.

Описание: цвет текста. Переопределяет значение, установленное системной переменной DIMCLRT. Возможные значения: см. свойство Color.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRotated 123ee2c4>
(vla-get-TextColor obj)
;;; 0
(vla-put-TextColor obj acRed)
```

8.2.552. Свойство *TextEditor*

Объект: **PreferencesFiles**.

Описание: имя текстового редактора, используемого командой MTEXT (МТЕКСТ). Возможные значения: строка. Не проверяется.

```
(setq obj (vla-get-Files
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesFiles2 05c00390>
(vla-get-TextEditor obj)
;;; "Internal"
(vla-put-TextEditor obj "Any string")
```

8.2.553. Свойство *TextFont*

Объект: **PreferencesDisplay**.

Описание: экранный шрифт командной строки. Возможные значения: строка.

```
(setq obj (vla-get-Display
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesDisplay 05c172dc>
(vla-get-TextFont obj)
;;; "Courier"
(vla-put-TextFont obj "Fixedsys")
```

8.2.554. Свойство *TextFontSize*

Объект: **PreferencesDisplay**.

Описание: размер экранного шрифта командной строки. Возможные значения: положительное целое, ограничения размеров накладываются шрифтом.

```
(setq obj (vla-get-Display
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesDisplay 05c172dc>
(vla-get-TextFontSize obj)
;;; 10
(vla-put-TextFontSize obj 12)
```

8.2.555. Свойство *TextFontStyle*

Объект: **PreferencesDisplay**.

Описание: стиль экранного шрифта командной строки. Возможные значения:

- `acFontRegular` (0) — обычный;
- `acFontItalic` (1) — курсив;

- acFontBold (2) — полужирный;
- acFontBoldItalic (3) — полужирный курсив.

```
(setq obj (vla-get-Display
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesDisplay 05c172dc>
(vla-get-TextFontStyle obj)
;;; 0
(vla-put-TextFontStyle obj acFontItalic)
```

8.2.556. Свойство *TextFrameDisplay*

Объект: **DatabasePreferences**.

Описание: показ рамки вместо текста. Соответствует системной переменной QTEXTMODE. Возможные значения:

- :vlax-true — показ рамки;
- :vlax-false — показ текста.

```
(setq obj (vla-get-Preferences actdoc))
;;; #<VLA-OBJECT IAcadDatabasePreferences 05c36024>
(vla-get-TextFrameDisplay obj)
;;; :vlax-false
(vla-put-TextFrameDisplay obj :vlax-true)
(vla-Regen actdoc acAllViewports)
```

8.2.557. Свойство *TextGap*

Объекты: **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**, **Leader**.

Описание: расстояние между текстом и размерной линией. Переопределяет значение, установленное системной переменной DIMGAP. Возможные значения: неотрицательное вещественное число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRotated 05c365d4>
(vla-get-TextGap obj)
;;; 0.625
(vla-put-TextGap obj 1.5)
```

8.2.558. Свойство *TextGenerationFlag*

Объекты: **Attribute**, **AttributeReference**, **Text**, **TextStyle**.

Описание: флаги генерации текста.

Возможные значения (могут суммироваться):

- 0 — нормальный (только чтение);
- `acTextFlagBackward` (2) — задом наперед;
- `acTextFlagUpsideDown` (4) — вверх ногами.

Примечание

Еще один "билет в один конец", возможность установки 0 отсутствует!

```
(setq obj (vlax-ename->vla-object (car (nentsel))))
;;; #<VLA-OBJECT IAcadAttributeReference 0b4ba144>
(vla-get-TextGenerationFlag obj)
;;; 0
(vla-put-TextGenerationFlag obj acTextFlagUpsideDown)
```

8.2.559. Свойство *TextHeight*

Объекты: **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**, **LabelTemplate**, **Tolerance**.

Описание: высота шрифта текста. Переопределяет значение, установленное системной переменной DIMTXT (кроме объекта `LabelTemplate`). Возможные значения: положительное вещественное число.

Примечание

Значение игнорируется, если текстовый стиль имеет фиксированную высоту.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRotated 0b4baaa4>
(vla-get-TextHeight obj)
;;; 2.5
(vla-put-TextHeight obj 4)
```

8.2.560. Свойство *TextInside*

Объекты: **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimDiametric**, **DimRadial**, **DimRadialLarge**, **DimRotated**.

Описание: принудительное размещение размерного текста между выносными линиями. Переопределяет значение, установленное системной переменной DIMTIX. Возможные значения:

- `:vlax-true` — включено;
- `:vlax-false` — выключено.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRotated 0b4baaa4>
(vla-get-TextInside obj)
;;; :vlax-false
(vla-put-TextInside obj :vlax-true)
```

8.2.561. Свойство *TextInsideAlign*

Объекты: **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimDiametric**, **DimRadial**, **DimRadialLarge**, **DimRotated**.

Описание: расположение текста относительно размерной линии (горизонтально или параллельно) для всех размеров, кроме ординатного. Переопределяет значение, установленное системной переменной DIMTIH. Возможные значения:

- `:vlax-true` — горизонтальное;
- `:vlax-false` — параллельное.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRotated 0b4baaa4>
(vla-get-TextInsideAlign obj)
;;; :vlax-false
(vla-put-TextInsideAlign obj :vlax-true)
```

8.2.562. Свойство *TextMovement*

Объекты: **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**.

Описание: изменение вида размера при перемещении размерного текста. Переопределяет значение, установленное системной переменной DIMTMOVE. Возможные значения:

- `acDimLineWithText` (0) — размерная линия следует за текстом;
- `acMoveTextAddLeader` (1) — текст перемещается без размерной линии, с добавлением выноски;
- `acMoveTextNoLeader` (2) — текст перемещается без размерной линии, без выноски.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRotated 0b4baaa4>
(vla-get-TextMovement obj)
;;; 0
(vla-put-TextMovement obj acMoveTextAddLeader)
```

8.2.563. Свойство *TextOutsideAlign*

Объекты: **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimDiametric**, **DimRadial**, **DimRadialLarge**, **DimRotated**.

Описание: горизонтальное расположение размерного текста вне выносных линий. Переопределяет значение, установленное системной переменной DIMTOH. Возможные значения:

- `:vlax-true` — включено;
- `:vlax-false` — выключено.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRotated 0b4baaa4>
(vla-get-TextOutsideAlign obj)
;;; :vlax-false
(vla-put-TextOutsideAlign obj :vlax-true)
```

8.2.564. Свойство *TextOverride*

Объекты: **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**.

Описание: строка переопределения размерного текста. Возможные значения: строка.

Примечание

Установка свойству пустой строки (`""`) приводит к восстановлению исходного размерного текста. Угловые скобки (`<>`) соответствуют основному размеру, квадратные скобки (`[]`) — альтернативному.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRotated 0b4baaa4>
(vla-get-TextOverride obj)
;;; ""
(vla-put-TextOverride obj "<>mm[]")
```

8.2.565. Свойство *TextPosition*

Объекты: **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**.

Описание: точка вставки размерного текста. Возможные значения: вариант (безопасный массив), содержащий координаты трехмерной точки.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRotated 0b4baaa4>
```

```
(setq temp (vla-get-TextPosition obj))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (1768.19 517.109 0.0)
(vla-put-TextPosition obj (vlax-3d-point (getpoint)))
```

8.2.566. Свойство *TextPrecision*

Объекты: **Dim3PointAngular**, **DimAngular**.

Описание: формат угловых единиц примитива. Переопределяет значение, установленное системной переменной DIMADEC. Возможные значения см. свойство `PrimaryUnitsPrecision`.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDim3PointAngular 0b763474>
(vla-get-TextPrecision obj)
;;; 0
(vla-put-TextPrecision obj acDimPrecisionThree)
```

8.2.567. Свойство *TextPrefix*

Объекты: **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**.

Описание: префикс размерного текста. Переопределяет значение, установленное системной переменной DIMPOST. Возможные значения: строка.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRotated 0b7605e4>
(vla-get-TextPrefix obj)
;;; ""
(vla-put-TextPrefix obj "Разм. ")
```

8.2.568. Свойство *TextRotation*

Объекты: **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**.

Описание: угол наклона размерного текста. Возможные значения: от 0 до 2π.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRotated 0b7605e4>
(vla-get-TextRotation obj)
;;; 0.0
(vla-put-TextRotation obj (/ pi 4))
```

8.2.569. Свойство *TextString*

Объекты: **Attribute**, **AttributeReference**, **MText**, **Text**, **Tolerance**.

Описание: строковое значение. Возможные значения: строка.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadText2 0b126aa4>
(vla-get-TextString obj)
;;; "Тестовая строка"
(vla-put-TextString obj "Любая строка")
```

8.2.570. Свойство *TextStyle*

Объекты: **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**, **LabelTemplate**, **Tolerance**.

Описание: стиль текста. Переопределяет значение, установленное системной переменной DIMTXSTY (кроме `LabelTemplate`). Возможные значения: строка (стиль должен существовать).

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRotated 0b126614>
(vla-get-TextStyle obj)
;;; "Standard"
(vla-put-TextStyle obj "MyTextStyle")
```

8.2.571. Свойство *TextStyles*

Объект: **Document**.

Описание: указатель на семейство текстовых стилей. Только чтение. Возможные значения: VLA-объект.

```
(vla-get-TextStyles actdoc)
;;; #<VLA-OBJECT IAcadTextStyles 0b7752b4>
```

8.2.572. Свойство *TextSuffix*

Объекты: **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**.

Описание: суффикс размерного текста. Переопределяет значение, установленное системной переменной DIMPOST. Возможные значения: строка.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRotated 0b7605e4>
```

```
(vla-get-TextSuffix obj)
;;; ""
(vla-put-TextSuffix obj " прим.")
```

8.2.573. Свойство *TextureMapPath*

Объект: **PreferencesFiles**.

Описание: путь поиска текстур. Возможные значения: строка (перечень путей). Не проверяется.

```
(setq obj (vla-get-Files
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesFiles2 0b775930>
(setq temp (vla-get-TextureMapPath obj))
;;; "D:\\Depl ... erDataCache\\Textures;D:\\Textures\\ADT"
(vla-put-TextureMapPath obj (strcat temp ";C:\\AddTextures"))
```

8.2.574. Свойство *TextWinBackgrndColor*

Объект: **PreferencesDisplay**.

Описание: цвет фона текстового экрана. Возможные значения: вариант типа OLE_COLOR.

Примечание

Этот тип данных не поддерживается AutoLISP. Тем не менее управлять цветом можно, зная цифровые коды цветов, но нельзя получить значение фактически установленного цвета.

```
(setq obj (vla-get-Display
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesDisplay 0b77470c>
(setq temp (vla-get-TextWinBackgrndColor obj))
;;; #<variant 19 16777215>
(vlax-variant-value temp)
;;; error: LISPFromPtr failed. The type is not supported: 19
(vla-put-TextWinBackgrndColor obj (vlax-make-variant 12634758 19))
```

8.2.575. Свойство *TextWinTextColor*

Объект: **PreferencesDisplay**.

Описание: цвет текста текстового экрана. Возможные значения: вариант типа OLE_COLOR.

Примечание

Этот тип данных не поддерживается в AutoLISP. Тем не менее управлять цветом можно, зная цифровые коды цветов, но нельзя получить значение фактически установленного цвета.

```
(setq obj (vla-get-Display
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesDisplay 0b77470c>
(setq temp (vla-get-TextWinTextColor obj))
;;; #<variant 19 0>
(vlax-variant-value temp)
;;; error: LISPFromPtr failed. The type is not supported: 19
(vla-put-TextWinTextColor obj (vlax-make-variant 12634758 19))
```

8.2.576. Свойство *Thickness*

Объекты: **Arc**, **Attribute**, **AttributeReference**, **Circle**, **LightweightPolyline**, **Line**, **Point**, **Polyline**, **Shape**, **Solid**, **Text**, **Trace**.

Описание: высота выдавливания плоского объекта. Возможные значения: вещественное число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadLine 0b774f44>
(vla-get-Thickness obj)
;;; 0.0
(vla-put-Thickness obj 200)
```

8.2.577. Свойство *TimeServer*

Объект: **SecurityParams**.

Описание: имя сервера времени. Возможные значения: строка, в том числе пустая.

```
(setq obj (vla-GetInterfaceObject
            (vlax-get-acad-object) "AutoCAD.SecurityParams.16"))
;;; #<VLA-OBJECT IAcadSecurityParams 0ba1cde4>
(vla-get-TimeServer obj)
;;; ""
(vla-put-TimeServer obj "Any string")
```

8.2.578. Свойство *TimeStamp*

Объект: **FileDependency**.

Описание: время создания внешнего файла в секундах, считая от 00:00:00 01.01.1980. Только чтение. Возможные значения: положительное целое число.

```
(setq obj (vla-Item (vla-get-FileDependencies actdoc) 1))
;;; #<VLA-OBJECT IAcadFileDependency 0ba20c64>
(vla-get-TimeStamp obj)
;;; 806794724
```

8.2.579. Свойство *TipOffset*

Объект: **LabelTemplate**.

Описание: смещение точки привязки текста. Возможные значения: координаты трехмерной точки в виде варианта, содержащего безопасный массив.

```
(setq DBConn (vla-GetInterfaceObject (vlax-get-acad-object)
               "CAO.DbConnect.16"))
;;; #<VLA-OBJECT IDbConnect 059d7b40>
(setq DBLablsT (vlax-invoke DBConn 'GetLabelTemplates))
;;; #<VLA-OBJECT ILabelTemplates 011593f0>
(setq DBLablT (vla-Item DBLablsT 0)) ;_был создан заранее
;;; #<VLA-OBJECT ILabelTemplate 01159a50>
(setq tmp (vlax-get-property DBLablT 'TipOffset))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value tmp))
;;; (0.0 0.0 0.0)
(vlax-put-property DBLablT 'TipOffset (vlax-3d-point '(1.0 1.5 0.0)))
```

8.2.580. Свойство *Title*

Объект: **SummaryInfo**.

Описание: заголовок документа. Возможные значения: строка, в том числе пустая.

```
(setq obj (vla-get-SummaryInfo actdoc))
;;; #<VLA-OBJECT IAcadSummaryInfo 0ba20d64>
(vla-get-Title obj)
;;; ""
(vla-put-Title obj "Любая строка")
```

8.2.581. Свойство *TitleSuppressed*

Объекты: **Table**, **TableStyle**.

Описание: режим подавления шапки. Возможные значения:

- `:vlax-true` — подавлять;
- `:vlax-false` — не подавлять.

```
(setq obj (vla-Item (vla-Item
     (vla-get-Dictionaries actdoc) "ACAD_TABLESTYLE") "Standard"))
;;; #<VLA-OBJECT IAcadTableStyle 0ba200f4>
(vla-get-TitleSuppressed obj)
;;; :vlax-false
(vla-put-TitleSuppressed obj :vlax-true)
```

8.2.582. Свойство *ToleranceDisplay*

Объекты: **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**.

Описание: отображение допусков. Переопределяет значение, установленное системной переменной DIMTOL. Возможные значения:

- `acTolNone` (0) — не отображать;
- `acTolSymmetrical` (1) — симметричный;
- `acTolDeviation` (2) — отклонение;
- `acTolLimits` (3) — предельные размеры;
- `acTolBasic` (4) — номинальный.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRotated 0ba6f0d4>
(vla-get-ToleranceDisplay obj)
;;; 0
(vla-put-ToleranceDisplay obj acTolLimits)
```

8.2.583. Свойство *ToleranceHeightScale*

Объекты: **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**.

Описание: масштабный коэффициент высоты текста допуска относительно основного размерного текста. Переопределяет значение, установленное системной переменной DIMTFAC. Возможные значения: положительное вещественное число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRotated 0ba6f0d4>
(vla-get-ToleranceHeightScale obj)
;;; 1.0
(vla-put-ToleranceHeightScale obj 0.625)
```

8.2.584. Свойство *ToleranceJustification*

Объекты: **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**.

Описание: вертикальное выравнивание допусков. Переопределяет значение, установленное системной переменной DIMTOLJ. Возможные значения:

- `acTolBottom` (0) — по низу;
- `acTolMiddle` (1) — посередине;
- `acTolTop` (2) — по верху.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRotated 0ba6f0d4>
(vla-get-ToleranceJustification obj)
;;; 0
(vla-put-ToleranceJustification obj acTolMiddle)
```

8.2.585. Свойство *ToleranceLowerLimit*

Объекты: **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**.

Описание: нижний предел допуска. Переопределяет значение, установленное системной переменной DIMTM. Возможные значения: положительное вещественное число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRotated 0ba6f0d4>
(vla-get-ToleranceLowerLimit obj)
;;; 1.0e-009
(vla-put-ToleranceLowerLimit obj 0.025)
```

8.2.586. Свойство *TolerancePrecision*

Объекты: **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**.

Описание: точность допусков. Переопределяет значение, установленное системной переменной DIMTDEC. Возможные значения см. свойство `PrimaryUnitsPrecision`.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRotated 0ba6f0d4>
(vla-get-TolerancePrecision obj)
;;; 2
(vla-put-TolerancePrecision obj acDimPrecisionThree)
```

8.2.587. Свойство *ToleranceSuppressLeadingZeros*

Объекты: **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**.

Описание: подавление ведущих нулей допусков. Переопределяет значение, установленное системной переменной DIMTZIN. Возможные значения:

- `:vlax-true` — включено;
- `:vlax-false` — выключено.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRotated 0ba6f0d4>
(vla-get-ToleranceSuppressLeadingZeros obj)
;;; :vlax-false
(vla-put-ToleranceSuppressLeadingZeros obj :vlax-true)
```

8.2.588. Свойство *ToleranceSuppressTrailingZeros*

Объекты: **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**.

Описание: подавление хвостовых нулей допусков. Переопределяет значение, установленное системной переменной DIMTZIN. Возможные значения:

- `:vlax-true` — включено;
- `:vlax-false` — выключено.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRotated 0ba6f0d4>
(vla-get-ToleranceSuppressTrailingZeros obj)
;;; :vlax-true
(vla-put-ToleranceSuppressTrailingZeros obj :vlax-false)
```

8.2.589. Свойство *ToleranceSuppressZeroFeet*

Объекты: **DimAligned**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**.

Описание: подавление "футовых" нулей допусков. Переопределяет значение, установленное системной переменной DIMTZIN. Возможные значения:

- `:vlax-true` — подавлять;
- `:vlax-false` — не подавлять.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRotated 0ba6f0d4>
```

```
(vla-get-ToleranceSuppressZeroFeet obj)
;;; :vlax-true
(vla-put-ToleranceSuppressZeroFeet obj :vlax-false)
```

8.2.590. Свойство *ToleranceSuppressZeroInches*

Объекты: **DimAligned**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**.

Описание: подавление "дюймовых" нулей допусков. Переопределяет значение, установленное системной переменной DIMTZIN. Возможные значения:

- `:vlax-true` — подавлять;
- `:vlax-false` — не подавлять.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRotated 0ba6f0d4>
(vla-get-ToleranceSuppressZeroInches obj)
;;; :vlax-true
(vla-put-ToleranceSuppressZeroInches obj :vlax-false)
```

8.2.591. Свойство *ToleranceUpperLimit*

Объекты: **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**.

Описание: верхний предел допуска. Переопределяет значение, установленное системной переменной DIMTP. Возможные значения: положительное вещественное число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRotated 0ba6f0d4>
(vla-get-ToleranceUpperLimit obj)
;;; 0.0
(vla-put-ToleranceUpperLimit obj 0.01)
```

8.2.592. Свойство *Toolbars*

Объект: **MenuGroup**.

Описание: указатель семейства панелей инструментов. Только чтение. Возможные значения: VLA-объект.

```
(setq obj (vla-Item (vla-get-MenuGroups (vlax-get-acad-object)) "acad"))
;;; #<VLA-OBJECT IAcadMenuGroup 010a9e54>
(vla-get-Toolbars obj)
;;; #<VLA-OBJECT IAcadToolbars 0ba796c4>
```

8.2.593. Свойство *ToolPalettePath*

Объект: **PreferencesFiles**.

Описание: путь к файлам хранения инструментальных палитр. Переопределяет значение, установленное системной переменной _TOOLPALETTEPATH. Возможные значения: строка.

Примечание

При установке несуществующего пути соответствующие папки создаются автоматически и в них помещаются необходимые файлы. Путь для создания папки должен быть возможен.

```
(setq obj (vla-get-Files
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesFiles2 0ba7b8a0>
(vla-get-ToolPalettePath obj)
;;; "C:\\Documents ... enu\\support\\ToolPalette"
(vla-put-ToolPalettePath obj "C:\\ToolPalette\\")
```

8.2.594. Свойство *Top*

Описание: расстояние от верхнего края экрана до верхнего края плавающей панели инструментов в пикселах. Возможные значения: неотрицательное целое число.

```
(setq obj (vla-Item (vla-get-Toolbars (vla-Item (vla-get-MenuGroups
        (vlax-get-acad-object)) "acad")) "Layers"))
;;; #<VLA-OBJECT IAcadToolbar 010d058c>
(vla-get-Top obj)
;;; 202
(vla-put-Top obj 400)
```

8.2.595. Свойство *TotalAngle*

Объект: **Arc**.

Описание: центральный угол в радианах. Только чтение. Возможные значения: от 0 до 2π.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadArc 055d96e4>
(vla-get-TotalAngle obj)
;;; 2.90349
```

8.2.596. Свойство *TranslateIDs*

Объект: **XRecord**.

Описание: преобразование ID объектов во время операций клонирования. Возможные значения:

- :vlax-true — да;
- :vlax-false — нет.

```
(defun demo (dname rname / dicts mydic myrec)
  (setq dicts (vla-get-Dictionaries
                (vla-get-ActiveDocument (vlax-get-acad-object))))
  (if (vl-catch-all-error-p (setq mydic (vl-catch-all-apply
        (function vla-Item) (list dicts dname))))
    (setq mydic (vla-Add dicts dname)))
  (if (vl-catch-all-error-p (setq myrec (vl-catch-all-apply
        (function vla-Item) (list mydic rname))))
    (setq myrec (vla-AddXRecord mydic rname)))
  myrec)
(setq obj (demo "MyTestDict" "MyTestRec"))
;;; #<VLA-OBJECT IAcadXRecord 055de7c4>
(vla-get-TranslateIDs obj)
;;; :vlax-true
(vla-put-TranslateIDs obj :vlax-false)
```

8.2.597. Свойство *Transparency*

Объект: **Raster**.

Описание: прозрачность растра (для некоторых типов изображений). Возможные значения:

- :vlax-true — прозрачный;
- :vlax-false — непрозрачный.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadRasterImage 055df964>
(vla-get-Transparency obj)
;;; :vlax-false
(vla-put-Transparency obj :vlax-true)
```

8.2.598. Свойство *TrueColor*

Объекты: **все графические объекты**, **Group**, **Layer**.

Описание: точный цвет. Возможные значения: VLA-объект.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadCircle 0ba8b584>
(vla-get-TrueColor obj)
;;; #<VLA-OBJECT IAcadAcCmColor 0ba8c320>
(setq temp (vla-GetInterfaceObject
    (vlax-get-acad-object) "AutoCAD.AcCmColor.16"))
;;; #<VLA-OBJECT IAcadAcCmColor 0ba884e0>
(vla-put-EntityColor temp -1036280892)
(vla-put-TrueColor obj temp)
```

8.2.599. Свойство *TrueColorImages*

Объект: **PreferencesDisplay**.

Описание: отображение растров в полной палитре цветов. Возможные значения:

- `:vlax-true` — включено;
- `:vlax-false` — выключено.

```
(setq obj (vla-get-Display
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesDisplay 0ba8f32c>
(vla-get-TrueColorImages obj)
;;; :vlax-true
(vla-put-TrueColorImages obj :vlax-false)
```

8.2.600. Свойство *TwistAngle*

Объект: **PViewport**.

Описание: угол поворота вида в радианах. Возможные значения: от 0 до 2π.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadPViewport2 0ba8b474>
(vla-get-TwistAngle obj)
;;; 0.0
(vla-put-TwistAngle obj (/ pi 4))
```

8.2.601. Свойство *Type*

Объекты: **3DPolyline** , **KeyDescription**, **Layout**, **Leader**, **MenuGroup**, **PlotConfiguration**, **Polyline**, **PolygonMesh**, **PopupMenuItem**, **ToolbarItem**.

Описание: тип. Возможные значения:

- для `3DPolyline`

  `acSimple3Dpoly` (0) — обычная;

  `acQuadSpline3Dpoly` (1) — квадратичный B-сплайн;

  `acCubicSpline3Dpoly` (2) — кубический B-сплайн.

- для `KeyDescription`

  `kCaoTypeUnknown` (0) — неизвестный;

  `kCaoTypeText` (1) — текстовый;

  `kCaoTypeNumeric` (2) — числовой;

  `kCaoTypeInteger` (3) — целочисленный;

  `kCaoTypeDouble` (4) — целочисленный длинный;

  `kCaoTypeDate` (5) — дата;

  `kCaoTypeTime` (6) — время;

  `kCaoTypeTimestamp` (7) — временная отметка;

  `kCaoTypeBit` (8) — логический;

  `kCaoTypeBlob` (9) — бинарные данные.

Примечание

Текущие версии Visual LISP константы этого типа не поддерживают. Изменение типа на недопустимый приводит к автоматическому сбросу значения на заданный соответствующему полю таблицы.

- для `Layout` и `PlotConfiguration`

  `acDisplay` (0) — печатать экран;

  `acExtents` (1) — печатать границы;

  `acLimits` (2) — печатать лимиты;

  `acView` (3) — печатать именованный вид;

  `acWindow` (4) — печатать окно;

  `acLayout` (5) — печатать вкладку.

- для `Leader`

  `acLineNoArrow` (0) — линейная, без стрелки;

  `acSplineNoArrow` (1) — сглаженная, без стрелки;

  `acLineWithArrow` (2) — линейная, со стрелкой;

  `acSplineWithArrow` (3) — сглаженная, со стрелкой.

- для `MenuGroup`

  `acBaseMenuGroup` (0) — базовое меню;

  `acPartialMenuGroup` (1) — фрагментное (подгружаемое) меню.

- для `Polyline`

  `acSimplePoly` (0) — обычная;

  `acFitCurvePoly` (1) — сглаженная;

`acQuadSplinePoly` (2) — квадратичный B-сплайн;

`acCubicSplinePoly` (3) — кубический B-сплайн.

- для `PolygonMesh`

  `acSimpleMesh` (0) — обычная полигональная сеть;

  `acQuadSurfaceMesh` (5) — квадратичный B-сплайн;

  `acCubicSurfaceMesh` (6) — кубический B-сплайн;

  `acBezierSurfaceMesh` (8) — поверхность Безье.

- для `PopupMenuItem`

  `acMenuItem` (0) — меню;

  `acMenuSeparator` (1) — разделитель;

  `acMenuSubMenu` (2) — подменю.

- для `ToolbarItem`

  `acToolbarButton` (0) — обычная кнопка;

  `acToolbarSeparator` (1) — разделитель;

  `acToolbarControl` (2) — раскрывающийся список;

  `acToolbarFlyout` (3) — групповая кнопка (подменю).

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadPolyline 0ba993b4>
(vla-get-Type obj)
;;; 3
(vla-put-Type obj acFitCurvePoly)
```

8.2.602. Свойство *UCSIconAtOrigin*

Объекты: **`PViewport`**, **`Viewport`**.

Описание: показ знака ПСК в начале координат. Переопределяет значение системной переменной UCSICON. Возможные значения:

- `:vlax-true` — включен;
- `:vlax-false` — выключен.

```
(setq obj (vla-get-ActivePViewport actdoc))
;;; #<VLA-OBJECT IAcadPViewport2 0ba8b474>
(vla-get-UCSIconAtOrigin obj)
;;; :vlax-true
(vla-put-UCSIconAtOrigin obj :vlax-false)
```

8.2.603. Свойство *UCSIconOn*

Объекты: **PViewport**, **Viewport**.

Описание: показ знака ПСК. Переопределяет значение системной переменной UCSICON. Возможные значения:

- `:vlax-true` — включен;
- `:vlax-false` — выключен.

```
(setq obj (vla-get-ActivePViewport actdoc))
;;; #<VLA-OBJECT IAcadPViewport2 0ba8b474>
(vla-get-UCSIconOn obj)
;;; :vlax-true
(vla-put-UCSIconOn obj :vlax-false)
```

8.2.604. Свойство *UCSPerViewport*

Объект: **PViewport**.

Описание: сохранение ПСК вместе с видовым экраном. Возможные значения:

- `:vlax-true` — включено;
- `:vlax-false` — выключено.

```
(setq obj (vla-get-ActivePViewport actdoc))
;;; #<VLA-OBJECT IAcadPViewport2 0ba8b474>
(vla-get-UCSPerViewport obj)
;;; :vlax-true
(vla-put-UCSPerViewport obj :vlax-false)
```

8.2.605. Свойство *Units*

Объект: **Block**.

Описание: единицы измерения, учитываемые при вставке. Возможные значения: см. свойство `ADCInsertUnitsDefaultSource`.

```
(setq obj (vla-Item (vla-get-Blocks actdoc)
    (vla-get-Name (vlax-ename->vla-object (car (entsel))))))
;;; #<VLA-OBJECT IAcadBlock3 02816694>
(vla-get-Units obj)
;;; 0
(vla-put-Units obj acInsertUnitsFeet)
```

8.2.606. Свойство *UnitsFormat*

Объекты: **DimAligned**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**.

Описание: формат линейных единиц измерения. Переопределяет значение, установленное системной переменной DIMLUNIT. Возможные значения:

- `acDimLScientific` (1) — научные;
- `acDimLDecimal` (2) — десятичные;
- `acDimLEngineering` (3) — инженерные;
- `acDimLArchitectural` (4) — архитектурные;
- `acDimLFractional` (5) — дробные;
- `acDimLWindowsDesktop` (6) — по региональным настройкам ОС.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRotated 0a11c084>
(vla-get-UnitsFormat obj)
;;; 2
(vla-put-UnitsFormat obj acDimLEngineering)
```

8.2.607. Свойство *UnitsType*

Объект: **DynamicBlockReferenceProperty**.

Описание: тип единиц измерения. Только чтение. Возможные значения:

- `acNoUnits` (0) — нет;
- `acAngular` (1) — угловые;
- `acDistance` (2) — линейные;
- `acArea` (3) — площадь.

```
(setq obj (vlax-safearray-get-element (vlax-variant-value
    (vla-GetDynamicBlockProperties (vlax-ename->vla-object
    (car (entsel))))) 0))
;;; #<VLA-OBJECT IAcadDynamicBlockReferenceProperty 02809484>
(vla-get-UnitsType obj)
;;; 2
```

8.2.608. Свойство *Updatable*

Объект: **Link**.

Описание: возможность обновления. Только чтение. Возможные значения:

- `:vlax-true` — да;
- `:vlax-false` — нет.

```
(setq DBConn (vla-GetInterfaceObject (vlax-get-acad-object)
               "CAO.DbConnect.16"))
;;; #<VLA-OBJECT IDbConnect 05725070>
```

```
(setq DBLinks (vlax-invoke DBConn 'GetLinks))
;;; #<VLA-OBJECT ILinks 05725570>
(setq DBLink (vla-Item DBLinks 0))
;;; #<VLA-OBJECT ILink 05723de0>
(vlax-get-property DBLink 'Updatable)
;;; :vlax-true
```

8.2.609. Свойство *UpperRightCorner*

Объект: **Viewport**.

Описание: координаты правого верхнего угла, в долях экрана. Только чтение. Возможные значения: вариант (безопасный массив), содержащий двумерные координаты.

```
(setq obj (vla-Item (vla-get-Viewports actdoc) 0))
;;; #<VLA-OBJECT IAcadViewport 0a165384>
(setq temp (vla-get-UpperRightCorner obj))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (1.0 1.0)
```

8.2.610. Свойство *UpsideDown*

Объекты: **Attribute**, **AttributeReference**, **Text**.

Описание: генерация текста "вверх ногами". Возможные значения:

- `:vlax-true` — включена;
- `:vlax-false` — выключена.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadText2 0a166fa4>
(vla-get-UpsideDown obj)
;;; :vlax-false
(vla-put-UpsideDown obj :vlax-true)
```

8.2.611. Свойство *URL*

Объект: **Hyperlink**.

Описание: адрес гиперссылки. Возможные значения: строка. Не проверяется.

```
(setq obj (vla-Item (vla-get-Hyperlinks
    (vlax-ename->vla-object (car (entsel)))) 0))
;;; #<VLA-OBJECT IAcadHyperlink 0a177d14>
(vla-get-URL obj)
;;; "..\\..\\ ... For WEB\\content\\clice.dwf"
(vla-put-URL obj "Any string")
```

8.2.612. Свойство *URLDescription*

Объект: **Hyperlink**.

Описание: пояснение к гиперссылке. Возможные значения: строка.

```
(setq obj (vla-Item (vla-get-Hyperlinks
    (vlax-ename->vla-object (car (entsel)))) 0))
;;; #<VLA-OBJECT IAcadHyperlink 0a177d14>
(vla-get-URLDescription obj)
;;; "Тестовая ссылка"
(vla-put-URLDescription obj "Any string")
```

8.2.613. Свойство *URLNamedLocation*

Объект: **Hyperlink**.

Описание: закладка в документе или вид в рисунке для объекта гиперссылки. Возможные значения: строка.

```
(setq obj (vla-Item (vla-get-Hyperlinks
    (vlax-ename->vla-object (car (entsel)))) 0))
;;; #<VLA-OBJECT IAcadHyperlink 0a177d14>
(vla-get-URLNamedLocation obj)
;;; ""
(vla-put-URLNamedLocation obj "Any string")
```

8.2.614. Свойство *Used*

2005

Объект: **Layer**.

Описание: признак того, что слой использован. Только чтение. Возможные значения:

- `:vlax-true` — да;
- `:vlax-false` — нет.

```
(setq obj (vla-Item (vla-get-Layers actdoc) "Layer3"))
;;; #<VLA-OBJECT IAcadLayer2 0a166bb4>
(vla-get-Used obj)
;;; :vlax-false
```

8.2.615. Свойство *UseLastPlotSettings*

Объект: **PreferencesOutput**.

Описание: признак использования в текущем выводе настроек предыдущей печати.

Возможные значения:

- `:vlax-true` — да;
- `:vlax-false` — нет.

```
(setq obj (vla-get-Output
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesOutput2 0a166a98>
(vla-get-UseLastPlotSettings obj)
;;; :vlax-true
(vla-put-UseLastPlotSettings obj :vlax-false)
```

8.2.616. Свойство *User*

Объект: **Preferences**.

Описание: указатель на объект, содержащий пользовательские настройки. Только чтение. Возможные значения: VLA-объект.

```
(setq obj (vla-get-Preferences (vlax-get-acad-object)))
;;; #<VLA-OBJECT IAcadPreferences 1387f56c>
(vla-get-User obj)
;;; #<VLA-OBJECT IAcadPreferencesUser 1387f568>
```

8.2.617. Свойство *UserCoordinateSystems*

Объект: **Document**.

Описание: указатель на семейство пользовательских систем координат документа. Только чтение. Возможные значения: VLA-объект.

```
(vla-get-UserCoordinateSystems actdoc)
;;; #<VLA-OBJECT IAcadUCSs 1387f2c4>
```

8.2.618. Свойство *UseStandardScale*

Объекты: **Layout**, **PlotConfiguration**.

Описание: использование стандартного масштаба печати. Возможные значения:

- `:vlax-true` — да;
- `:vlax-false` — нет.

```
(setq obj (vla-get-ActiveLayout actdoc))
;;; #<VLA-OBJECT IAcadLayout 138942d4>
(vla-get-UseStandardScale obj)
;;; :vlax-true
(vla-put-UseStandardScale obj :vlax-false)
```

8.2.619. Свойство *Utility*

Объект: **Document**.

Описание: указатель на семейство утилит. Только чтение. Возможные значения: VLA-объект.

```
(vla-get-Utility actdoc)
;;; #<VLA-OBJECT IAcadUtility 13894ea4>
```

8.2.620. Свойство *Value*

Объекты: **DynamicBlockReferenceProperty**, **IDPair**, **KeyValue**.

Описание: значение (value), для `IDPair` — ID нового объекта. Для `IDPair` и `KeyValue` только чтение. Возможные значения: вариант, содержащий строку, для `IDPair` — положительное целое.

```
(setq DBConn (vla-GetInterfaceObject (vlax-get-acad-object)
               "CAO.DbConnect.16"))
;;; #<VLA-OBJECT IDbConnect 0573b0f0>
(setq DBLinks (vlax-invoke DBConn 'GetLinks))
;;; #<VLA-OBJECT ILinks 0573b430>
(setq DBLink (vla-Item DBLinks 0))
;;; #<VLA-OBJECT ILink 0573e840>
(setq KeyVals (vlax-get-property DBLink 'KeyValues))
;;; #<VLA-OBJECT IKeyValues 13b14980>
(setq obj (vla-Item KeyVals 0))
;;; #<VLA-OBJECT IKeyValue 13b139b0>
(setq temp (vla-get-Value obj))
;;; #<variant 8 6021>
(vlax-variant-value temp)
;;; "6021"
```

8.2.621. Свойство *VBE*

Объект: **Application**.

Описание: указатель приложения VBA IDE. Только чтение. Возможные значения: VLA-объект.

```
(setq obj (vlax-get-acad-object))
;;; #<VLA-OBJECT IAcadApplication 00b9d604>
(vla-get-VBE obj)
;;; #<VLA-OBJECT VBE 18ed1704>
```

8.2.622. Свойство *Verify*

Объекты: **Attribute**, **AttributeReference**.

Описание: признак проверяемости атрибута (установка только для Attribute). Возможные значения:

- :vlax-true — включен;
- :vlax-false — выключен.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadAttribute 0573a524>
(vla-get-Verify obj)
;;; :vlax-false
(vla-put-Verify obj :vlax-true)
```

8.2.623. Свойство *Version*

Объекты: **Application**, **DbConnect**.

Описание: версия. Только чтение. Возможные значения: строка.

```
(setq obj (vlax-get-acad-object))
;;; #<VLA-OBJECT IAcadApplication 00b9d604>
(vla-get-Version obj)
;;; "16.1s (LMS Tech)"
(setq DBConn (vla-GetInterfaceObject (vlax-get-acad-object)
               "CAO.DbConnect.16"))
;;; #<VLA-OBJECT IDbConnect 05725070>
(vla-get-Version DBConn)
;;; "AutoCAD SQL Environment T"
```

8.2.624. Свойство *VersionGuid*

Объект: **FileDependency**.

Описание: глобально уникальный идентификатор (GUID) версии. Возможные значения: строка вида "{307EDEA7-A83A-449D-A37E-12ECB7533D3E}".

```
(setq obj (vla-Item (vla-get-FileDependencies actdoc) 1))
;;; #<VLA-OBJECT IAcadFileDependency 1e68bc24>
(vla-get-VersionGuid obj)
;;; "{307EDEA7-A83A-449D-A37E-12ECB7533D3E}"
```

8.2.625. Свойство *VertCellMargin* 2005

Объекты: **Table**, **TableStyle**.

Описание: вертикальные поля ячеек. Возможные значения: неотрицательное вещественное число.

Примечание

При необходимости ячейки растягиваются, но не сжимаются (сдвигается текст).

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadTable 1e68a4f4>
(vla-get-VertCellMargin obj)
;;; 1.5
(vla-put-VertCellMargin obj 5)
```

8.2.626. Свойство *VerticalTextPosition*

Объекты: **Dim3PointAngular**, **DimAligned**, **DimAngular**, **DimArcLength**, **DimDiametric**, **DimOrdinate**, **DimRadial**, **DimRadialLarge**, **DimRotated**, **Leader**.

Описание: положение текста по вертикали относительно размерной линии. Переопределяет значение, установленное системной переменной DIMTAD. Возможные значения:

- acVertCentered (0) — по центру в разрыве размерной линии;
- acAbove (1) — над размерной линией;
- acOutside (2) — рядом с размерной линией со стороны, наиболее удаленной от определяющих точек;
- acJIS (3) — в соответствии с японскими промышленными стандартами (JIS).

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadDimRotated 1e697074>
(vla-get-VerticalTextPosition obj)
;;; 1
(vla-put-VerticalTextPosition obj acJIS)
```

8.2.627. Свойство *ViewportDefault*

Объект: **Layer**.

Описание: автоматическая заморозка в новых видовых экранах пространства листа. Возможные значения:

- :vlax-true — да;
- :vlax-false — нет.

```
(setq obj (vla-Item (vla-get-Layers actdoc) "Layer1"))
;;; #<VLA-OBJECT IAcadLayer2 1e6906f4>
(vla-get-ViewportDefault obj)
;;; :vlax-false
(vla-put-ViewportDefault obj :vlax-true)
```

8.2.628. Свойство *ViewportOn*

Объект: **PViewport**.

Описание: признак включенности. Возможные значения:

- :vlax-true — да;
- :vlax-false — нет.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadPViewport2 1e694734>
(vla-get-ViewportOn obj)
;;; :vlax-true
(vla-put-ViewportOn obj :vlax-false)
```

8.2.629. Свойство *Viewports*

Объект: **Document**.

Описание: указатель на семейство неперекрывающихся видовых экранов. Только чтение. Возможные значения: VLA-объект.

```
(vla-get-Viewports actdoc)
;;; #<VLA-OBJECT IAcadViewports 1e1ea6a4>
```

8.2.630. Свойство *Views*

Объект: **Document**.

Описание: указатель семейства видов. Только чтение. Возможные значения: VLA-объект.

```
(vla-get-Views actdoc)
;;; #<VLA-OBJECT IAcadViews 1e1ea4c4>
```

8.2.631. Свойство *ViewToPlot*

Объекты: **Layout**, **PlotConfiguration**.

Описание: имя печатаемого вида. Возможные значения: строка.

Примечание

Вид с данным именем должен быть заранее создан. Установка возможна только для вкладки модели.

```
(setq obj (vla-get-ActiveLayout actdoc))
;;; #<VLA-OBJECT IAcadLayout 1e1ea2d4>
(vla-get-ViewToPlot obj)
;;; ""
(vla-put-ViewToPlot obj "OutPutView")
```

8.2.632. Свойство *VisibilityEdge1*

Объект: **3DFace**.

Описание: видимость первой кромки. Возможные значения:

- `:vlax-true` — да;
- `:vlax-false` — нет.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcad3DFace 1e630954>
(vla-get-VisibilityEdge1 obj)
;;; :vlax-true
(vla-put-VisibilityEdge1 obj :vlax-false)
```

8.2.633. Свойство *VisibilityEdge2*

Объект: **3DFace**.

Описание: видимость второй кромки. Возможные значения:

- `:vlax-true` — да;
- `:vlax-false` — нет.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcad3DFace 1e630954>
(vla-get-VisibilityEdge2 obj)
;;; :vlax-false
(vla-put-VisibilityEdge2 obj :vlax-true)
```

8.2.634. Свойство *VisibilityEdge3*

Объект: **3DFace**.

Описание: видимость третьей кромки. Возможные значения:

- `:vlax-true` — да;
- `:vlax-false` — нет.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcad3DFace 1e630954>
(vla-get-VisibilityEdge3 obj)
;;; :vlax-true
(vla-put-VisibilityEdge3 obj :vlax-false)
```

8.2.635. Свойство *VisibilityEdge4*

Объект: **3DFace**.

Описание: видимость четвертой кромки.

Возможные значения:

- `:vlax-true` — да;
- `:vlax-false` — нет.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcad3DFace 1e630954>
(vla-get-VisibilityEdge4 obj)
;;; :vlax-false
(vla-put-VisibilityEdge4 obj :vlax-true)
```

8.2.636. Свойство *Visible*

Объекты: **все графические примитивы**, **Application**, **AttributeReference**, **Group**, **Toolbar**.

Описание: видимость объекта. Возможные значения:

- `:vlax-true` — видимый;
- `:vlax-false` — невидимый.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcad3DFace 0c172dc4>
(vla-get-Visible obj)
;;; :vlax-true
(vla-put-Visible obj :vlax-false)
```

8.2.637. Свойство *Volume*

Объект: **3DSolid**.

Описание: объем. Только чтение. Возможные значения: положительное число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcad3DSolid 0c173bf4>
(vla-get-Volume obj)
;;; 2.49118e+006
```

8.2.638. Свойство *Weights*

Объект: **Spline**.

Описание: относительный вес точек притяжения. Возможные значения: вариант (безопасный массив), содержащий вещественные числа.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadSpline 0c1b62b4>
(vla-SetWeight obj 0 1)
```

```
(setq temp (vla-get-Weights obj))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (1.0 1.0 1.0 1.0 1.0 1.0 1.0)
(vla-put-Weights obj (vlax-safearray-fill (vlax-make-safearray
    vlax-vbDouble '(0 . 6)) '(0.3 0.9 1.5 0.8 0.2 0.6 0.9)))
```

8.2.639. Свойство *Width*

Объекты: **Application**, **MText**, **PViewport**, **Raster**, **Table**, **TextStyle**, **Toolbar**, **Viewport**, **View**.

Описание: ширина объекта (установка, кроме `Toolbar` и `Raster`). Возможные значения: неотрицательное вещественное число (кроме `Application` и `Toolbar`, для которых — целое).

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadMText2 0c1931b4>
(vla-get-Width obj)
;;; 93.382
(vla-put-Width obj 110)
```

8.2.640. Свойство *WindowLeft*

Объект: **Application**.

Описание: положение левой границы окна приложения. Возможные значения: положительное целое число.

```
(setq obj (vlax-get-acad-object))
;;; #<VLA-OBJECT IAcadApplication 00b9d604>
(vla-get-WindowLeft obj)
;;; 42
(vla-put-WindowLeft obj 60)
```

8.2.641. Свойство *WindowState*

Объекты: **Application**, **Document**.

Описание: состояние окна. Возможные значения:

- `acNorm` (1) — развернуто;
- `acMin` (2) — свернуто;
- `acMax` (3) — максимизировано.

```
(vla-get-WindowState actdoc)
;;; 3
(vla-put-WindowState actdoc acMin)
```

8.2.642. Свойство *WindowTitle*

Объект: **Document**.

Описание: заголовок. Только чтение. Возможные значения: строка.

```
(vla-get-WindowTitle actdoc)
;;; "D:\\Tutor\\Draw\\Tutorial.dwg"
```

8.2.643. Свойство *WindowTop*

Объект: **Application**.

Описание: положение верхней границы окна приложения. Возможные значения: положительное целое число.

```
(setq obj (vlax-get-acad-object))
;;; #<VLA-OBJECT IAcadApplication 00b9d604>
(vla-get-WindowTop obj)
;;; 27
(vla-put-WindowTop obj 60)
```

8.2.644. Свойство *WorkspacePath*

Объект: **PreferencesFiles**.

Описание: путь к UDL-файлам подключаемых баз данных. Возможные значения: любая строка.

```
(setq obj (vla-get-Files
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesFiles2 0c196db0>
(vla-get-WorkspacePath obj)
;;; "C:\\Documents ... enu\\Data Links"
(vla-put-WorkspacePath obj "C:\\Temp\\")
```

8.2.645. Свойство *XEffectiveScaleFactor* 2006

Объекты: **BlockRef**, **ExternalReference**, **MInsertBlock**.

Описание: фактический масштаб по оси *X*. Возможные значения: ненулевое вещественное число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadBlockReference2 028266c4>
(vla-get-XEffectiveScaleFactor obj)
;;; 1.3
(vla-put-XEffectiveScaleFactor obj 2)
```

8.2.646. Свойство *XRefDatabase*

Объект: **Block**.

Описание: указатель на базу документа внешней ссылки. Только чтение. Возможные значения: VLA-объект.

```
(setq obj (vla-Item (vla-get-Blocks actdoc)
    (vla-get-Name (vlax-ename->vla-object (car (entsel))))))
;;; #<VLA-OBJECT IAcadBlock2 0c0ed3a4>
(vla-get-XRefDatabase obj)
;;; #<VLA-OBJECT IAcadDatabase 0c0ed7f4>
```

8.2.647. Свойство *XrefDemandLoad*

Объект: **PreferencesOpenSave**.

Описание: режим подгрузки внешних ссылок. Соответствует системной переменной XLOADCTL. Возможные значения:

- `acDemandLoadDisabled` (0) — подгрузка запрещена;
- `acDemandLoadEnabled` (1) — разрешено использовать оригинал;
- `acDemandLoadEnabledWithCopy` (2) — загружается копия.

```
(setq obj (vla-get-OpenSave
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesOpenSave 0c0f1b14>
(vla-get-XrefDemandLoad obj)
;;; 0
(vla-put-XrefDemandLoad obj acDemandLoadEnabled)
```

8.2.648. Свойство *XRefEdit*

Объект: **DatabasePreferences**.

Описание: доступность документа для редактирования его как внешней ссылки из другого рисунка. Соответствует системной переменной XEDIT. Возможные значения:

- `:vlax-true` — разрешает;
- `:vlax-false` — запрещает.

```
(setq obj (vla-get-Preferences actdoc))
;;; #<VLA-OBJECT IAcadDatabasePreferences 0c0f8d94>
(vla-get-XRefEdit obj)
;;; :vlax-true
(vla-put-XRefEdit obj :vlax-false)
```

8.2.649. Свойство *XRefFadeIntensity*

Объект: **PreferencesDisplay**.

Описание: интенсивность отображения объектов при прямом редактировании вхождений внешних ссылок. Соответствует системной переменной XFADECTL. Возможные значения: целое число от 1 до 90.

```
(setq obj (vla-get-Display
            (vla-get-Preferences (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadPreferencesDisplay 0c0e8b5c>
(vla-get-XRefFadeIntensity obj)
;;; 50
(vla-put-XRefFadeIntensity obj 30)
```

8.2.650. Свойство *XRefLayerVisibility*

Объект: **DatabasePreferences**.

Описание: сохранение установок свойств слоев внешних ссылок независимо от ссылок. Соответствует системной переменной VISRETAIN. Возможные значения:

- `:vlax-true` — да;
- `:vlax-false` — нет.

```
(setq obj (vla-get-Preferences actdoc))
;;; #<VLA-OBJECT IAcadDatabasePreferences 0c0f8d94>
(vla-get-XRefLayerVisibility obj)
;;; :vlax-true
(vla-put-XRefLayerVisibility obj :vlax-false)
```

8.2.651. Свойство *XScaleFactor*

Объекты: **BlockRef**, **ExternalReference**, **MInsertBlock**.

Описание: масштабный коэффициент по оси *X*. Возможные значения: вещественное число, кроме нуля.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadBlockReference 0c11afa4>
(vla-get-XScaleFactor obj)
;;; 1.0
(vla-put-XScaleFactor obj 1.5)
```

8.2.652. Свойство *XVector*

Объект: **UCS**.

Описание: направление оси *X*. Соответствует системной переменной UCSXDIR. Возможные значения: вариант (безопасный массив), содержащий координаты по трем осям.

Примечание

В свойствах объекта вектор направления хранится в виде координат мнимой точки на поверхности единичной сферы, при этом устанавливать можно произвольные координаты, которые будут спроецированы на единичную сферу автоматически.

```
(setq obj (vla-Item (vla-get-UserCoordinateSystems actdoc) "Test"))
;;; #<VLA-OBJECT IAcadUCS 0c10d194>
(setq temp (vla-get-XVector obj))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (0.976012 0.217715 0.0)
(vla-put-XVector obj (vlax-3d-point '(0.9 1.6 0.3)))
```

8.2.653. Свойство *YEffectiveScaleFactor*

Объекты: **BlockRef**, **ExternalReference**, **MInsertBlock**.

Описание: фактический масштаб по оси *Y*. Возможные значения: ненулевое вещественное число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadBlockReference2 028266c4>
(vla-get-YEffectiveScaleFactor obj)
;;; 1.4
(vla-put-YEffectiveScaleFactor obj 2)
```

8.2.654. Свойство *YScaleFactor*

Объекты: **BlockRef**, **ExternalReference**, **MInsertBlock**.

Описание: масштабный коэффициент по оси *Y*. Возможные значения: вещественное число, кроме нуля.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadBlockReference 0c11afa4>
(vla-get-YScaleFactor obj)
;;; 1.0
(vla-put-YScaleFactor obj 1.2)
```

8.2.655. Свойство *YVector*

Объект: **UCS**.

Описание: направление оси *Y*. Соответствует системной переменной UCSYDIR. Возможные значения: вариант (безопасный массив), содержащий координаты по трем осям.

Примечание

В свойствах объекта вектор направления хранится в виде координат мнимой точки на поверхности единичной сферы, при этом устанавливать можно произвольные координаты, которые будут спроецированы на единичную сферу автоматически.

```
(setq obj (vla-Item (vla-get-UserCoordinateSystems actdoc) "Test"))
;;; #<VLA-OBJECT IAcadUCS 0c10d194>
(setq temp (vla-get-YVector obj))
;;; #<variant 8197 ...>
(vlax-safearray->list (vlax-variant-value temp))
;;; (-0.192508 0.86301 0.467071)
(vla-put-YVector obj (vlax-3d-point '(1.9 -11.6 -5.3)))
```

8.2.656. Свойство *ZEffectiveScaleFactor*

Объекты: **BlockRef**, **ExternalReference**, **MInsertBlock**.

Описание: фактический масштаб по оси *Z*. Возможные значения: ненулевое вещественное число.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadBlockReference2 028266c4>
(vla-get-ZEffectiveScaleFactor obj)
;;; 1.5
(vla-put-ZEffectiveScaleFactor obj 2)
```

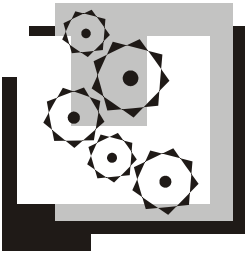
8.2.657. Свойство *ZScaleFactor*

Объекты: **BlockRef**, **ExternalReference**, **MInsertBlock**.

Описание: масштабный коэффициент по оси *Z*. Возможные значения: вещественное число, кроме нуля.

```
(setq obj (vlax-ename->vla-object (car (entsel))))
;;; #<VLA-OBJECT IAcadBlockReference 0c11afa4>
(vla-get-ZScaleFactor obj)
;;; 1.0
(vla-put-ZScaleFactor obj 3
```

ГЛАВА 9



События и реакторы

Помимо объектов, свойств и методов, объектная модель оперирует еще и событиями. *События* — это системные уведомления об изменении программной обстановки: действиях пользователя с объектами системы (графическими примитивами, падающими меню, панелями инструментов и т. д.), устройствами (мышью, клавиатурой и др.).

Технология ActiveX позволяет создавать особые объекты, называемые *реакторами*, которые дают возможность спроектировать реакцию системы на происходящие события. Реакторы еще называются *VLR-объектами*.

9.1. Обрабатываемые события

События — это своеобразные элементы объектной модели, созданной и описанной разработчиками системы. Поэтому система AutoCAD умеет реагировать и предоставляет пользователю возможность создавать другие реакции только на те события, которые были сформулированы заранее.

В табл. 9.1 приведен перечень основных событий, заложенных в объектную модель AutoCAD версий 2005 и 2006. События, наименования которых выделены заглавными буквами и начинаются с префиксов `ACSM_`, `CALLOUT_`, `MODEL_` и `SHEET_`, связаны с подшивками листов и в настоящее время (в версии 2006) не могут обрабатываться на языке LISP.

Таблица 9.1. *Перечень событий*

Событие	Описание
`ACSM_DATABASE_CHANGED`	База изменена
`ACSM_DATABASE_CREATED`	База создана
`ACSM_DATABASE_IS_CLOSING`	База будет закрыта

Таблица 9.1 *(продолжение)*

Событие	Описание
ACSM_DATABASE_LOAD_BEGIN	Начата загрузка базы
ACSM_DATABASE_LOAD_COMPLETE	Окончена загрузка базы
ACSM_DATABASE_LOAD_FAILED	Загрузка базы завершилась аварийно
ACSM_DATABASE_LOCKED	База заблокирована
ACSM_DATABASE_OPENED	База открыта
ACSM_DATABASE_UNLOCKED	База разблокирована
Activate	Активировано окно документа
AppActivate	Получен запрос на активизацию главного окна приложения
AppDeactivate	Получен запрос на деактивацию главного окна приложения
ARXLoaded	Загружено ARX-приложение
ARXUnloaded	Выгружено ARX-приложение
BeginClose	Получен запрос на закрытие рисунка
BeginCommand	Выполнение команды началось
BeginDocClose	Получен запрос на закрытие рисунка (с возможностью воспрепятствовать закрытию)
BeginDoubleClick	Выполнен двойной щелчок на объекте
BeginFileDrop	Начата загрузка файла в главное окно приложения
BeginLISP	Получен запрос на вычисление LISP-выражения
BeginModal	Будет загружено модальное окно
BeginOpen	Получен запрос на открытие файла существующего рисунка
BeginPlot	Получен запрос на печать рисунка
BeginQuit	Будет закрыт документ или завершен сеанс системы AutoCAD
BeginRightClick	Начат щелчок правой кнопкой мыши в графическом экране
BeginSave	Получен запрос на сохранение рисунка
BeginShortcutMenuCommand	Закончен щелчок правой кнопкой мыши в графическом экране и ожидается вывод контекстного меню в режиме работы команды
BeginShortcutMenuDefault	Закончен щелчок правой кнопкой мыши в графическом экране и ожидается вывод контекстного меню в режиме работы по умолчанию

Таблица 9.1 *(продолжение)*

Событие	**Описание**
BeginShortcutMenuEdit	Закончен щелчок правой кнопкой мыши в графическом экране и ожидается вывод контекстного меню в режиме редактирования
BeginShortcutMenuGrip	Закончен щелчок правой кнопкой мыши в графическом экране и ожидается вывод контекстного меню в режиме ручек
BeginShortcutMenuOsnap	Закончен щелчок правой кнопкой мыши в графическом экране и ожидается вывод контекстного меню в режиме объектной привязки
CALLOUT_BLOCK_ADDED	Блок-идентификатор добавлен
CALLOUT_BLOCK_DELETED	Блок-идентификатор удален
Deactivate	Окно деактивировано
EndCommand	Команда завершена
EndLISP	Закончено вычисление LISP-выражения
EndModal	Модальное окно закрыто
EndOpen	Существующий файл рисунка открыт
EndPlot	Документ отправлен на принтер
EndSave	Сохранение рисунка закончено
EndShortcutMenu	Контекстное меню закрылось
LayoutSwitched	Активизирована другая вкладка
LISPCancelled	Вычисление LISP-выражения прервано
MODEL_RESOURCE_ADDED	Ресурс добавлен
MODEL_RESOURCE_DELETED	Ресурс удален
Modified	Изменен объект или семейство
NewDrawing	Получен запрос на создание нового рисунка
ObjectAdded	К рисунку добавлен объект
ObjectErased	Объект удален из рисунка
ObjectModified	Объект изменен
OnConnect	Соединение установлено
OnDisconnect	Соединение разорвано
OnLabelTemplateCreated	Шаблон ярлыка создан
OnLabelTemplateDeleted	Шаблон ярлыка удален
OnLabelTemplateModified	Шаблон ярлыка изменен

Таблица 9.1 *(окончание)*

Событие	Описание
OnLinkTemplateCreated	Шаблон связи создан
OnLinkTemplateDeleted	Шаблон связи удален
OnLinkTemplateModified	Шаблон связи изменен
OnQueryCreated	Запрос создан
OnQueryDeleted	Запрос удален
OnQueryModified	Запрос изменен
SelectionChanged	Изменен набор предварительно выбранных объектов
SHEET_ADDED	Лист добавлен
SHEET_DELETED	Лист удален
SHEET_SELECTION_SET_CREATED	Набор листов выбран
SHEET_SELECTION_SET_DELETED	Набор листов удален
SHEET_SUBSET_CREATED	Группа листов создана
SHEET_SUBSET_DELETED	Группа листов удалена
SHEET_VIEW_CATEGORY_CREATED	Категория видов создана
SHEET_VIEW_CATEGORY_DELETED	Категория видов удалена
SHEET_VIEW_CREATED	Вид создан
SHEET_VIEW_DELETED	Вид удален
SHEETS_RENUMBERED	Листы перенумерованы
SysVarChanged	Изменено значение системной переменной
WindowChanged	Изменено окно приложения или документа
WindowMovedOrResized	Окно приложения или документа перемещено или изменены размеры окна

Для обозначения многих событий применяются специальные зарезервированные символы (например, `:vlr-sysVarChanged`). Эти символы будут перечисляться при описании функций, создающих реакторы соответствующих типов.

На большую часть событий можно создать специальный объект (реактор), который по получению соответствующего уведомления будет запускать программу-обработчик события.

9.2. Классификация реакторов

Каждый реактор знает имена своих событий и действия, которые он должен выполнить. *Действия (реакции)* являются ничем иным, как функциями на языке

LISP (примерно такими же, какие программировались с помощью функции `action_tile` в диалоговых окнах).

Функции, имеющие префикс `vlr-`, работают с реакторами: создают реакторы, активизируют, отключают и т. п. Для того чтобы эти функции были доступны, следует выполнить функцию `vl-load-com`, которая загружает расширение языка AutoLISP. В ранних версиях вместо функции `vl-load-com` использовалась функция `vl-load-reactors`.

Основные типы реакторов и константы, зарезервированные для их обозначения, приведены в табл. 9.2.

Таблица 9.2. *Основные типы реакторов*

Обозначение	**Описание**
`:VLR-AcDb-Reactor`	Реакторы базы
`:VLR-DocManager-Reactor`	Реакторы документа
`:VLR-Linker-Reactor`	Реакторы связи
`:VLR-Object-Reactor`	Реакторы объектов
`:VLR-Editor-Reactor`	Реакторы редактирования (сохранены только для совместимости с предыдущими версиями)

Реакторы базы отслеживают события изменения базы объектов рисунка. Реакторы документа учитывают действия, связанные с документами (открытие, активизация, блокирование и т. п.). Реакторы связи предназначены для операций загрузки и выгрузки ARX-приложений. Реакторы объектов следят за состоянием конкретных объектов рисунка.

Реакторы редактирования сохранены в системе AutoCAD только для совместимости с предыдущими версиями. Они могут быть заменены реакторами, типы которых перечислены в табл. 9.3.

Таблица 9.3. *Типы реакторов редактирования*

Обозначение	**Описание**
`:VLR-Command-Reactor`	Реакторы команд
`:VLR-DeepClone-Reactor`	Реакторы клонирования
`:VLR-DWG-Reactor`	DWG-реакторы
`:VLR-DXF-Reactor`	DXF-реакторы
`:VLR-Insert-Reactor`	Реакторы команды INSERT (ВСТАВИТЬ)
`:VLR-Lisp-Reactor`	Реакторы LISP-событий
`:VLR-Miscellaneous-Reactor`	Реакторы других событий редактирования

Таблица 9.3 *(окончание)*

Обозначение	Описание
:VLR-Mouse-Reactor	Реакторы действий мышью
:VLR-SysVar-Reactor	Реакторы изменения системных переменных
:VLR-Toolbar-Reactor	Реакторы изменения размеров кнопок панелей инструментов
:VLR-Undo-Reactor	Реакторы команды UNDO (ОТМЕНИТЬ)
:VLR-Wblock-Recator	Реакторы команды WBLOCK (ПБЛОК)
:VLR-Window-Reactor	Реакторы изменения формы и размеров окна системы AutoCAD
:VLR-XREF-Reactor	Реакторы событий вставки и редактирования внешних ссылок

9.3. Справочные функции реакторов

В этом разделе приведены функции общего назначения, позволяющие получать справки о реакторах.

9.3.1. Функция *vlr-types*

Выводит список всех типов реакторов.

```
(vlr-types)
```

Возвращаемое значение — список с зарезервированными символами, обозначающими типы реакторов (константы приведены в табл. 9.2 и 9.3).

9.3.2. Функция *vlr-reactors*

Выводит список всех реакторов или реакторов заданного типа, загруженных в данный рисунок.

```
(vlr-reactors [<тип1> [<тип2> ... [<типN>]...]])
```

Аргументы — символы из табл. 9.2 и 9.3 с зарезервированными именами типов реакторов. Если аргументы опущены, то выводится список реакторов всех типов.

Возвращаемое значение — список, элементами которого являются списки. Внутренние списки имеют следующий вид: `(<тип> <реактор> <реактор> ...)`, например:

```
(:VLR-Object-Reactor #<VLR-Object-Reactor> #<VLR-Object-Reactor>)
```

Первый элемент этого списка — зарезервированный символ, остальные — VLR-объекты (указатели). Из списка можно извлечь нужные реакторы (VLR-объекты).

Если реакторов нужного типа в рисунке нет, то возвращается `nil`.

9.3.3. Функция *vlr-type*

Вычисляет тип реактора.

```
(vlr-type <реактор>)
```

Аргумент `<реактор>` — VLR-объект.

Возвращаемое значение — зарезервированный символ с обозначением типа реактора.

9.4. Действия реакторов

Каждый реактор отвечает только на те события (callback events), на которые он спроектирован. Перечень событий, требующих реакции, зависит от типа реактора и может быть получен с помощью функции `vlr-reaction-names` *(см. разд. 9.4.1)*.

Функции действия (callback function) реакторов создаются как обычные функции (с помощью функции `defun`), но с некоторыми **ограничениями** *(см. также правила, изложенные в разд. 9.8)*:

- нельзя применять функции `command` и vl-cmdf;
- при работе с объектами рисунка следует использовать не функции, обращающиеся к примитивам (как `entmod` и `entget`), а соответствующие функции ActiveX (`vla-put-StartPoint` и т. п.).

Функции действия должны иметь **два аргумента** (исключение для реакторов объектов):

- VLR-объект (указатель на реактор, который вызвал функцию действия);
- список параметров, формируемый системой AutoCAD.

Функции действия **реакторов объектов** должны иметь **три аргумента**. Первым идет VLA-объект (владелец), а затем VLR-объект и список параметров, формируемый системой.

Структура списка параметров зависит от порождающего события и приводится ниже в разделах, описывающих функции создания реакторов соответствующих типов.

Функция действия должна быть вместе с именем события при создании реактора включена в точечную пару *(см. разд. 9.5)*.

В системе AutoCAD предусмотрены две функции (`vlr-beep-reaction` и `vlr-trace-reaction`), которые могут быть использованы в качестве функций действия при написании и отладке реакторов.

9.4.1. Функция *vlr-reaction-names*

Выводит список событий, отслеживаемых реакторами данного типа.

```
(vlr-reaction-names <тип>)
```

Аргумент <тип> — символ из числа зарезервированных для обозначения типов реакторов (см. табл. 9.2 и 9.3).

Возвращаемое значение — список с символами, зарезервированными для обозначения событий.

Пример:

- `(vlr-reaction-names :VLR-Undo-Reactor)` возвращает
  `(:VLR-undoSubcommandAuto :VLR-undoSubcommandControl`
  `:VLR-undoSubcommandBegin :VLR-undoSubcommandEnd`
  `:VLR-undoSubcommandMark :VLR-undoSubcommandBack`
  `:VLR-undoSubcommandNumber)`

9.4.2. Функция *vlr-beep-reaction*

Выдает звуковой сигнал.

```
(vlr-beep-reaction [<аргументы>])
```

Может использоваться как функция действия при отладке реакторов. Количество передаваемых ей аргументов может быть любым (их тип зависит от реактора).

Возвращаемое значение — `nil`.

9.4.3. Функция *vlr-trace-reaction*

Выводит значения аргументов, переданных реактору, в окно **Trace** (Трассировка) среды Visual LISP.

```
(vlr-trace-reaction [<аргументы>])
```

Тоже может использоваться как функция действия при отладке реакторов. Количество и типы передаваемых ей аргументов могут быть любыми.

Возвращаемое значение — `nil`.

См. пример функции `vlr-reaction-set` *(разд. 9.5.20)*.

9.5. Создание реакторов

Для того чтобы создать реактор (VLR-объект), необходимо связать событие с некоторыми данными и прикладной функцией, которая будет выполняться с этими данными при наступлении указанного события (этот принцип не распространяется только на реакторы типа `:VLR-Object-Reactor`, т. е. реакторы объектов). Создание реактора нужного типа выполняется с помощью функции, которая имеет имя, практически совпадающее с обозначением типа реактора, приведенным в табл. 9.2 и 9.3, например: реактор типа `:VLR-Linker-Reactor` (реактор ARX-приложений) создается функцией `vlr-linker-reactor`. Функции создания реакторов всех типов рассмотрены ниже.

Для реакторов каждого типа есть свой перечень событий (он приводится вместе с описанием функции, создающей реакторы данного типа).

Иначе организованы реакторы объектов. Реактор объектов связывается в первую очередь с VLA-объектами рисунка, которые называются *владельцами* (owners) этого реактора. Он соединяет объекты и события с данными и прикладной функцией. Реакторы объектов создаются функцией `vlr-object-reactor`.

Для замены компонентов реактора может применяться функция `vlr-reaction-set` (см. далее).

9.5.1. Функция *vlr-acdb-reactor*

Создает реактор типа `:VLR-AcDb-Reactor` (событий изменения базы рисунка — добавления, модификации и удаления примитивов).

```
(vlr-acdb-reactor <данные> <реакции>)
```

Аргументы: `<данные>` — значение любого типа (число, список, строка и т. д.), применяемого в LISP (`nil`, если данных нет); `<реакции>` — список из точечных пар с именами событий и функций действия.

Возвращаемое значение — VLR-объект (реактор).

Точечные пары в аргументе `<реакции>` состоят из двух символов и имеют вид:

```
(<событие> . <функция>)
```

Например

```
(:vlr-objectAppended . myrfun)
```

Зарезервированные имена событий для реакторов этого типа должны выбираться из табл. 9.4. Имена функций действия являются именами функций, создаваемых пользователями.

Помимо аргумента с именем вызывающего реактора функции действия в данном случае передается список только из одного аргумента — примитива, который вызвал изменение базы рисунка. О функциях действия см. также *разд. 9.4*.

Таблица 9.4. *События, используемые реакторами типа* `:VLR-AcDb-Reactor`

Событие	**Описание**
`:vlr-objectAppended`	Примитив добавлен
`:vlr-objectUnAppended`	Примитив стерт (например, с помощью команды UNDO (ОТМЕНИТЬ))
`:vlr-objectReAppended`	Примитив восстановлен (например, с помощью команды REDO (ПОВТОРИТЬ))
`:vlr-objectOpenedForModify`	Примитив будет модифицироваться

Таблица 9.4 *(окончание)*

Событие	Описание
:vlr-objectModified	Примитив изменен
:vlr-objectErased	Примитив помечен как удаленный
:vlr-objectUnErased	У примитива снята пометка удаления

9.5.2. Функция *vlr-docmanager-reactor*

Создает реактор типа :VLR-DocManager-Reactor (событий изменения рисунка как документа).

```
(vlr-docmanager-reactor <данные> <реакции>)
```

Аргументы: <данные> — значение любого типа, применяемого в AutoLISP (nil, если данных нет); <реакции> — список из точечных пар с именами событий и функций действия.

Имена событий для реакторов этого типа должны выбираться из табл. 9.5.

Помимо аргумента с именем вызывающего реактора в данном случае функции действия передается аргумент — список, элементы которого перечислены в табл. 9.6.

Возвращаемое значение — VLR-объект (реактор).

Таблица 9.5. *События, используемые реакторами типа* :VLR-DocManager-Reactor

Событие	Описание
:vlr-documentCreated	Создан новый документ (открыт новый рисунок или существующий)
:vlr-documentToBeDestroyed	Документ будет уничтожен
:vlr-documentLockModeWillChange	Команда должна начать или закончить модификацию элементов документа, собирается блокировать документ или снять блокировку
:vlr-documentLockModeChangeVetoed	Реактор наложил вето на себя из функции действия события :vlr-documentLockModeChanged
:vlr-documentLockModeChanged	Устанавливается блокировка документа или снимается
:vlr-documentBecameCurrent	Документ изменен (не означает, что документ стал активным)
:vlr-documentToBeActivated	Документ будет активизирован
:vlr-documentToBeDeactivated	Активизировано другое окно (может быть, даже вне AutoCAD)

В табл. 9.6 описаны аргументы, которые передаются функции действия реактора документов для разных событий в списке параметров, формируемом системой AutoCAD.

Таблица 9.6. *Параметры, передаваемые в списке функции действия реактора типа* `:VLR-DocManager-Reactor`

Событие	**Кол-во пара-метров**	**Параметры**
`:vlr-documentCreated`, `:vlr-documentToBeDestroyed`, `:vlr-documentBecameCurrent`, `:vlr-documentToBeActivated`, `:vlr-documentToBeDeactivated`	1	VLA-объект (документ)
`:vlr-documentLockModeChangeVetoed`	2	1 — VLA-объект документа; 2 — командная строка, передаваемая при запросе на блокировку (если производится запрос на разблокировку, то строка начинается с символа #)
`:vlr-documentLockModeWillChange`, `:vlr-documentLockModeChanged`	5	1 — VLA-объект документа; 2 — целое число, указывающее код текущего состояния блокировки; 3 — целое число, указывающее код состояния блокировки после события; 4 — наивысший код блокировки других происходящих событий; 5 — командная строка, передаваемая при запросе на блокировку (если производится запрос на разблокировку, то строка начинается с символа #). Код состояния блокировки может принимать следующие значения: 1 — автоматическая запись блокирована; 2 — нет блокировки; 4 — разделенная запись; 8 — чтение; 10 — исключительная запись

9.5.3. Функция *vlr-linker-reactor*

Создает реактор типа `:VLR-Linker-Reactor` (событий загрузки и выгрузки ARX-приложений).

```
(vlr-linker-reactor <данные> <реакции>)
```

Аргументы: `<данные>` — значение любого типа, применяемого в AutoLISP (`nil`, если данных нет); `<реакции>` — список из точечных пар с именами событий и функций действия.

Имена событий для реакторов этого типа должны выбираться из табл. 9.7. В списке параметров, формируемом системой AutoCAD и передаваемом функции действия реакторов рассматриваемого типа, должен быть только один аргумент — строка с именем приложения.

Возвращаемое значение — VLR-объект (реактор).

Таблица 9.7. *События, используемые реакторами типа* `:VLR-Linker-Reactor`

Событие	Описание
`:vlr-rxAppLoaded`	Загружено новое приложение ObjectARX, которое закончило инициализацию
`:vlr-rxAppUnLoaded`	Динамический загрузчик выгрузил ARX-приложение, и память освобождена

9.5.4. Функция *vlr-object-reactor*

Создает реактор типа `:VLR-Object-Reactor` (реактор объектов). Реактор добавляется к рисунку, но не становится постоянным.

```
(vlr-object-reactor <владельцы> <данные> <реакции>)
```

Аргументы: `<владельцы>` — список VLA-объектов, за которыми наблюдает реактор; `<данные>` — значение любого типа, применяемого в AutoLISP (`nil`, если данных нет); `<реакции>` — список из точечных пар с именами событий и функций действия.

Имена событий для реакторов этого типа должны выбираться из табл. 9.8.

Возвращаемое значение — VLR-объект (реактор), вызвавший функцию действия.

Помимо аргумента с именем вызывающего реактора в данном случае функции действия передается аргумент — список, элементы которого перечислены в табл. 9.9.

Таблица 9.8. *События, используемые реакторами типа* `:VLR-Object-Reactor`

Событие	Описание
`:vlr-cancelled`	Модификация объекта отменена
`:vlr-copied`	Объект скопирован
`:vlr-erased`	У объекта установлен признак удаленного примитива
`:vlr-unerased`	У объекта снят признак удаленного примитива
`:vlr-goodbye`	Объект будет удален из памяти
`:vlr-openedForModify`	Объект открыт для модификации

Таблица 9.8 *(окончание)*

Событие	Описание
`:vlr-modified`	Объект изменен. Если модификация отменена, то произойдут еще события `:vlr-cancelled` и `:vlr-modifyUndone`
`:vlr-subObjModified`	Изменен подпримитив объекта (например, вершина полилинии или сети или атрибут вхождения блока)
`:vlr-modifyUndone`	Отменена модификация объекта
`:vlr-modifiedXData`	Изменены расширенные данные объекта
`:vlr-unappended`	Объект отсоединен от базы рисунка
`:vlr-reappended`	Объект снова присоединен к базе рисунка
`:vlr-objectClosed`	Объект закрыт для модификации

Таблица 9.9. *Параметры, передаваемые в списке функции действия реактора типа* `:VLR-Object-Reactor`

Событие	Кол-во пара-метров	Параметры
`:vlr-cancelled`, `:vlr-erased`, `:vlr-unerased`, `:vlr-goodbye`, `:vlr-openedForModify`, `:vlr-modified`, `:vlr-modifyUndone`, `:vlr-modifiedXData`, `:vlr-unappended`, `:vlr-reappended`, `:vlr-objectClosed`	0	—
`:vlr-copied`	1	Объект, созданный командой копирования примитивов
`:vlr-subObjModified`	1	Измененный подпримитив

Пример:

- `(setq myVLR (vlr-object-reactor (list line1) "Реактор отрезка" '((:vlr-modified . showlen))))` создает реактор объекта `line1` (отрезка, являющегося VLA-объектом); реактор отслеживает события изменения отрезка и выполняет функцию `showlen` (например, она выводит новую длину)

9.5.5. Функция *vlr-editor-reactor*

Создает реактор типа `:VLR-Editor-Reactor` (реактор редактирования). Функция и данный тип реактора сохранены для совместимости с предыдущими версиями (предпочтительнее пользоваться реакторами, перечисленными в табл. 9.3).

```
(vlr-editor-reactor <данные> <реакции>)
```

Аргументы: <данные> — значение любого типа, применяемого в AutoLISP (nil, если данных нет); <реакции> — список из точечных пар с именами событий и функций действия.

Имена событий для реакторов этого типа должны выбираться из табл. 9.10.

Возвращаемое значение — VLR-объект (реактор).

Помимо аргумента с именем вызывающего реактора в данном случае функции действия передается аргумент — список, элементы которого перечислены в табл. 9.11.

Таблица 9.10. *События, используемые реакторами типа* `:VLR-Editor-Reactor`

Событие	Описание
:vlr-beginClose	База рисунка будет закрыта
:vlr-beginDxfIn	Содержимое DXF-файла будет добавлено к базе рисунка
:vlr-abortDxfIn	Операция импорта DXF-файла завершилась аварийно
:vlr-dxfInComplete	Импорт DXF-файла завершен успешно
:vlr-beginDxfOut	Начинается экспорт рисунка в DXF-файл
:vlr-abortDxfOut	Операция экспорта в DXF-файл завершилась аварийно
:vlr-dxfOutComplete	Операция экспорта в DXF-файл завершена успешно
:vlr-databaseToBeDestroyed	Содержимое базы рисунка будет удалено из памяти
:vlr-unknownCommand	Введена команда, не известная системе AutoCAD
:vlr-commandWillStart	Вызвана команда системы AutoCAD
:vlr-commandEnded	Команда закончена
:vlr-commandCancelled	Команда прервана
:vlr-commandFailed	Команда не смогла завершить работу
:vlr-lispWillStart	Начинается вычисление LISP-выражения
:vlr-lispEnded	Закончилось вычисление LISP-выражения
:vlr-lispCancelled	Вычисление LISP-выражения было прервано
:vlr-beginDwgOpen	Будет открыт рисунок
:vlr-endDwgOpen	Операция открытия рисунка закончена
:vlr-dwgFileOpened	Новый рисунок загружен
:vlr-beginSave	Начинается операция сохранения рисунка
:vlr-saveComplete	Завершена операция сохранения рисунка

Таблица 9.10 *(окончание)*

Событие	Описание
`:vlr-sysVarWillChange`	Начинается операция изменения значения системной переменной
`:vlr-sysVarChanged`	Изменено значение системной переменной

Таблица 9.11. *Параметры, передаваемые в списке функции действия реактора типа* `:VLR-Editor-Reactor`

Событие	Кол-во пара-метров	Параметры
`:vlr-lispEnded`, `:vlr-lispCancelled`, `:vlr-beginClose`, `:vlr-beginDxfIn`, `:vlr-abortDxfIn`, `:vlr-dxfInComplete`, `:vlr-beginDxfOut`, `:vlr-abortDxfOut`, `:vlr-dxfOutComplete`, `:vlr-databaseToBeDestroyed`	0	—
`:vlr-unknownCommand`, `:vlr-commandWillStart`, `:vlr-commandEnded`, `:vlr-commandCancelled`, `:vlr-commandFailed`	1	Строка с именем команды
`:vlr-lispWillStart`	1	Первая строка (начало LISP-выражения)
`:vlr-beginDwgOpen`, `:vlr-endDwgOpen`, `:vlr-dwgFileOpened`	1	Строка с именем открываемого файла
`:vlr-beginSave`	1	Строка с именем файла для сохранения, по умолчанию (может быть изменена пользователем)
`:vlr-saveComplete`	1	Строка с именем файла, в который выполнено сохранение
`:vlr-sysVarWillChange`	1	Строка с именем системной переменной
`:vlr-sysVarChanged`	2	1 — строка с именем системной переменной, 2 — целое число с признаком завершения операции (1 — изменение выполнено успешно, 0 — изменение завершилось аварийно)

9.5.6. Функция *vlr-command-reactor*

Создает реактор типа `:VLR-Command-Reactor` (реактор команд).

```
(vlr-command-reactor <данные> <реакции>)
```

Аргументы: `<данные>` — значение любого типа, применяемого в AutoLISP (`nil`, если данных нет); `<реакции>` — список из точечных пар с именами событий и функций действия.

Имена событий для реакторов этого типа должны выбираться из табл. 9.12. В списке параметров, формируемом системой AutoCAD и передаваемом функции действия реакторов рассматриваемого типа, должен быть только один аргумент — строка с именем приложения.

Возвращаемое значение — VLR-объект (реактор).

Таблица 9.12. *События, используемые реакторами типа* `:VLR-Command-Reactor`

Событие	Описание
`:vlr-unknownCommand`	Введена команда, не известная системе AutoCAD
`:vlr-commandWillStart`	Вызвана команда системы AutoCAD
`:vlr-commandEnded`	Команда закончена
`:vlr-commandCancelled`	Команда прервана
`:vlr-commandFailed`	Команда не смогла завершить работу

9.5.7. Функция *vlr-deepclone-reactor*

Создает реактор типа `:VLR-DeepClone-Reactor` (реактор событий клонирования объектов, т. е. переноса в другой файл).

```
(vlr-deepclone-reactor <данные> <реакции>)
```

Аргументы: `<данные>` — значение любого типа, применяемого в AutoLISP (`nil`, если данных нет); `<реакции>` — список из точечных пар с именами событий и функций действия.

Имена событий для реакторов этого типа должны выбираться из табл. 9.13.

Возвращаемое значение — VLR-объект (реактор).

Помимо аргумента с именем вызывающего реактора в данном случае функции действия передается аргумент — список, элементы которого перечислены в табл. 9.14.

Таблица 9.13. *События, используемые реакторами типа* `:VLR-DeepClone-Reactor`

Событие	Описание
`:vlr-beginDeepClone`	Начало операции клонирования
`:vlr-beginDeepCloneXlation`	Событие, происходящее между стадиями копирования объектов и трансляцией их ID (идентификаторов) в базе
`:vlr-abortDeepClone`	Операция клонирования прервана
`:vlr-endDeepClone`	Операция клонирования завершена

Таблица 9.14. *Параметры, передаваемые в списке функции действия реактора типа* `:VLR-DeepClone-Reactor`

Событие	Кол-во параметров	Параметры
`:vlr-beginDeepClone`, `:vlr-abortDeepClone`, `:vlr-endDeepClone`	0	—
`:vlr-beginDeepCloneXlation`	1	Целое число с кодом возврата (если это число соответствует коду ошибки, то операция клонирования прекращается)

9.5.8. Функция *vlr-dwg-reactor*

Создает реактор типа `:VLR-DWG-Reactor` (реактор событий, связанных с операциями над DWG-файлами).

```
(vlr-dwg-reactor <данные> <реакции>)
```

Аргументы: `<данные>` — значение любого типа, применяемого в AutoLISP (`nil`, если данных нет); `<реакции>` — список из точечных пар с именами событий и функций действия.

Имена событий для реакторов этого типа должны выбираться из табл. 9.15.

Возвращаемое значение — VLR-объект (реактор).

Помимо аргумента с именем вызывающего реактора в данном случае функции действия передается аргумент — список, элементы которого перечислены в табл. 9.16.

Таблица 9.15. *События, используемые реакторами типа* `:VLR-DWG-Reactor`

Событие	Описание
`:vlr-beginClose`	База рисунка будет закрыта
`:vlr-databaseConstructed`	Создана база рисунка
`:vlr-databaseToBeDestroyed`	Содержимое базы рисунка будет удалено из памяти
`:vlr-beginDwgOpen`	Будет открыт DWG-файл
`:vlr-endDwgOpen`	Операция открытия DWG-файла закончена
`:vlr-dwgFileOpened`	Новый рисунок загружен в окно системы AutoCAD
`:vlr-beginSave`	Файл будет сохранен
`:vlr-saveComplete`	Файл сохранен

Таблица 9.16. *Параметры, передаваемые в списке функции действия реактора типа* `:VLR-DWG-Reactor`

Событие	Кол-во пара-метров	Параметры
`:vlr-beginClose`, `:vlr-databaseConstructed`, `:vlr-databaseToBeDestroyed`	0	—
`:vlr-beginDwgOpen`, `:vlr-endDwgOpen`, `:vlr-dwgFileOpened`	1	Строка с именем файла
`:vlr-beginSave`	1	Строка с именем файла, предлагаемым для сохранения по умолчанию
`:vlr-saveComplete`	1	Строка с фактическим именем файла, в который выполнено сохранение

9.5.9. Функция *vlr-dxf-reactor*

Создает реактор типа `:VLR-DXF-Reactor` (реактор событий, связанных с операциями над DXF-файлами).

```
(vlr-dxf-reactor <данные> <реакции>)
```

Аргументы: `<данные>` — значение любого типа, применяемого в AutoLISP (`nil`, если данных нет); `<реакции>` — список из точечных пар с именами событий и функций действия.

Имена событий для реакторов этого типа должны выбираться из табл. 9.17.

Возвращаемое значение — VLR-объект (реактор).

Помимо аргумента с именем вызывающего реактора в данном случае функции действия передается аргумент — пустой список.

Таблица 9.17. *События, используемые реакторами типа* `:VLR-DXF-Reactor`

Событие	Описание
`:vlr-beginDxfIn`	Содержимое DXF-файла будет добавлено к базе рисунка
`:vlr-abortDxfIn`	Операция импорта DXF-файла завершилась аварийно
`:vlr-dxfInComplete`	Импорт DXF-файла завершен успешно
`:vlr-beginDxfOut`	Начинается экспорт рисунка в DXF-файл
`:vlr-abortDxfOut`	Операция экспорта в DXF-файл завершилась аварийно
`:vlr-dxfOutComplete`	Операция экспорта в DXF-файл завершена успешно

9.5.10. Функция *vlr-insert-reactor*

Создает реактор типа `:VLR-Insert-Reactor` (реактор событий, связанных со вставкой блоков).

```
(vlr-insert-reactor <данные> <реакции>)
```

Аргументы: `<данные>` — значение любого типа, применяемого в AutoLISP (`nil`, если данных нет); `<реакции>` — список из точечных пар с именами событий и функций действия.

Имена событий для реакторов этого типа должны выбираться из табл. 9.18.

Возвращаемое значение — VLR-объект (реактор).

Помимо аргумента с именем вызывающего реактора в данном случае функции действия передается аргумент — список, элементы которого перечислены в табл. 9.19.

Таблица 9.18. *События, используемые реакторами типа* `:VLR-Insert-Reactor`

Событие	Описание
`:vlr-beginInsert`	Начало операции вставки блока
`:vlr-beginInsertM`	К базе рисунка будет добавлена матрица трехмерного преобразования
`:vlr-otherInsert`	Блок или матрица добавлены в рисунок, но ID новых объектов вхождений блоков еще не транслированы
`:vlr-endInsert`	Конец вставки. Объекты уже добавлены к базе, но пока не могут использоваться в наборах (будут доступны после события `:vlr-commandEnded`)

Таблица 9.18 *(окончание)*

Событие	Описание
`:vlr-abortInsert`	Операция вставки блока прервана (база рисунка находится в нестабильном состоянии)

Таблица 9.19. *Параметры, передаваемые в списке функции действия реактора типа* `:VLR-Insert-Reactor`

Событие	Кол-во параметров	Параметры
`:vlr-beginInsert`	3	1 — VLA-объект (указатель базы, в которую вставляется блок); 2 — строка с именем блока; 3 — VLA-объект (указатель базы, из которой вставляется блок)
`:vlr-beginInsertM`	3	1 — VLA-объект (указатель базы, в которую вставляется матрица преобразования); 2 — матрица; 3 — VLA-объект (указатель базы, из которой вставляется матрица)
`:vlr-otherInsert`	2	1 — VLA-объект (указатель базы, в которую вставляется блок); 2 — VLA-объект (указатель базы, из которой вставляется блок)
`:vlr-endInsert`, `:vlr-abortInsert`	1	VLA-объект (указатель базы, в которую вставляется блок)

9.5.11. Функция *vlr-lisp-reactor*

Создает реактор типа `:VLR-Lisp-Reactor` (реактор событий, связанных с вводом и вычислением LISP-выражений).

```
(vlr-lisp-reactor <данные> <реакции>)
```

Аргументы: `<данные>` — значение любого типа, применяемого в AutoLISP (`nil`, если данных нет); `<реакции>` — список из точечных пар с именами событий и функций действия.

Имена событий для реакторов этого типа должны выбираться из табл. 9.20.

Возвращаемое значение — VLR-объект (реактор).

Помимо аргумента с именем вызывающего реактора в данном случае функции действия передается аргумент — список, элементы которого перечислены в табл. 9.21.

Таблица 9.20. *События, используемые реакторами типа* `:VLR-Lisp-Reactor`

Событие	Описание
`:vlr-lispWillStart`	Начинается ввод LISP-выражения
`:vlr-lispEnded`	Ввод и вычисление LISP-выражения закончено
`:vlr-lispCancelled`	Ввод LISP-выражения прерван

Таблица 9.21. *Параметры, передаваемые в списке функции действия реактора типа* `:VLR-Lisp-Reactor`

Событие	Кол-во параметров	Параметры
`:vlr-lispEnded`, `:vlr-lispCancelled`	0	—
`:vlr-lispWillStart`	1	Первая строка LISP-выражения

9.5.12. Функция *vlr-miscellaneous-reactor*

Создает реактор типа `:VLR-Miscellaneous-Reactor` (реактор прочих событий, работу которого не прерывают другие реакторы редактирования).

```
(vlr-miscellaneous-reactor <данные> <реакции>)
```

Аргументы: `<данные>` — значение любого типа, применяемого в AutoLISP (`nil`, если данных нет); `<реакции>` — список из точечных пар с именами событий и функций действия.

Имена событий для реакторов этого типа должны выбираться из табл. 9.22.

Возвращаемое значение — VLR-объект (реактор).

Помимо аргумента с именем вызывающего реактора в данном случае функции действия передается аргумент — список, элементы которого перечислены в табл. 9.23.

Таблица 9.22. *События, используемые реакторами типа* `:VLR-Miscellaneous-Reactor`

Событие	Описание
`:vlr-pickfirstModified`	Набор предварительного выбора изменен
`:vlr-layoutSwitched`	Выполнен переход на вкладку другого листа

Таблица 9.23. *Параметры, передаваемые в списке функции действия реактора типа* `:VLR-Miscellaneous-Reactor`

Событие	Кол-во параметров	Параметры
`:vlr-pickfirstModified`	0	—
`:vlr-layoutSwitched`	1	Строка с именем вкладки листа

9.5.13. Функция *vlr-mouse-reactor*

Создает реактор типа `:VLR-Mouse-Reactor` (реактор событий, связанных с мышью).

```
(vlr-mouse-reactor <данные> <реакции>)
```

Аргументы: `<данные>` — значение любого типа, применяемого в AutoLISP (`nil`, если данных нет); `<реакции>` — список из точечных пар с именами событий и функций действия.

Имена событий для реакторов этого типа должны выбираться из табл. 9.24.

Возвращаемое значение — VLR-объект (реактор).

Помимо аргумента с именем вызывающего реактора в данном случае функции действия передается аргумент — список с координатами трехмерной точки в МСК.

Таблица 9.24. *События, используемые реакторами типа* `:VLR-Mouse-Reactor`

Событие	Описание
`:vlr-beginDoubleClick`	Выполнен двойной щелчок
`:vlr-beginRightClick`	Выполнен щелчок правой кнопкой

9.5.14. Функция *vlr-sysvar-reactor*

Создает реактор типа `:VLR-SysVar-Reactor` (реактор событий, связанных с изменением значения системных переменных).

```
(vlr-sysvar-reactor <данные> <реакции>)
```

Аргументы: `<данные>` — значение любого типа, применяемого в AutoLISP (`nil`, если данных нет); `<реакции>` — список из точечных пар с именами событий и функций действия.

Имена событий для реакторов этого типа должны выбираться из табл. 9.25.

Возвращаемое значение — VLR-объект (реактор).

Помимо аргумента с именем вызывающего реактора в данном случае функции действия передается аргумент — список, элементы которого перечислены в табл. 9.26.

Таблица 9.25. *События, используемые реакторами типа* `:VLR-SysVar-Reactor`

Событие	Описание
`:vlr-sysVarWillChange`	Началась операция изменения значения системной переменной
`:vlr-sysVarChanged`	Значение системной переменной изменено

Таблица 9.26. *Параметры, передаваемые в списке функции действия реактора типа* `:VLR-SysVar-Reactor`

Событие	Кол-во параметров	Параметры
`:vlr-sysVarWillChange`	1	Строка с именем системной переменной
`:vlr-sysVarChanged`	2	1 — строка с именем системной переменной; 2 — признак завершения операции изменения системной переменной (`T` при успешном завершении, `nil` — при неуспешном)

9.5.15. Функция *vlr-toolbar-reactor*

Создает реактор типа `:VLR-Toolbar-Reactor` (реактор событий, связанных с изменением размера кнопок панелей инструментов).

```
(vlr-toolbar-reactor <данные> <реакции>)
```

Аргументы: `<данные>` — значение любого типа, применяемого в AutoLISP (`nil`, если данных нет); `<реакции>` — список из точечных пар с именами событий и функций действия.

Имена событий для реакторов этого типа должны выбираться из табл. 9.27.

Возвращаемое значение — VLR-объект (реактор).

Таблица 9.27. *События, используемые реакторами типа* `:VLR-Toolbar-Reactor`

Событие	Описание
`:vlr-toolbarBitmapSizeWillChange`	Размер кнопок панели будет изменен
`:vlr-toolbarBitmapSizeChanged`	Размер кнопок изменен

Функции действия в качестве списка параметров, формируемого системой AutoCAD, будет передаваться список из одного значения: `T`, если выбран большой размер кнопок, и `nil` — если выбран малый размер.

9.5.16. Функция *vlr-undo-reactor*

Создает реактор типа `:VLR-Undo-Reactor` (реактор событий, связанных с командой UNDO (ОТМЕНИТЬ)).

```
(vlr-undo-reactor <данные> <реакции>)
```

Аргументы: `<данные>` — значение любого типа, применяемого в AutoLISP (`nil`, если данных нет); `<реакции>` — список из точечных пар с именами событий и функций действия.

Имена событий для реакторов этого типа должны выбираться из табл. 9.28.

Возвращаемое значение — VLR-объект (реактор).

Помимо аргумента с именем вызывающего реактора в данном случае функции действия передается аргумент — список, элементы которого перечислены в табл. 9.29.

Таблица 9.28. *События, используемые реакторами типа* `:VLR-Undo-Reactor`

Событие	Описание
`:vlr-undoSubcommandAuto`	Команда UNDO (ОТМЕНИТЬ) выполняется с опцией **Auto** (Авто)
`:vlr-undoSubcommandControl`	Команда UNDO (ОТМЕНИТЬ) выполняется с опцией **Control** (Управление)
`:vlr-undoSubcommandBegin`	Команда UNDO (ОТМЕНИТЬ) выполняется с опциями **Begin** (Начало) или **Group** (Группа) (установка начального маркера команд, которые могут быть отменены как единое целое)
`:vlr-undoSubcommandEnd`	Команда UNDO (ОТМЕНИТЬ) выполняется с опцией **End** (Конец) (установка конечного маркера команд, которые могут быть отменены как единое целое)
`:vlr-undoSubcommandMark`	Команда UNDO (ОТМЕНИТЬ) выполняется с опцией **Mark** (Метка) (установка маркера для отмены команд с помощью опции **Back** (Назад) команды UNDO (ОТМЕНИТЬ))
`:vlr-undoSubcommandBack`	Команда UNDO (ОТМЕНИТЬ) выполняется с опцией **Back** (Назад) (отменяет все команды вплоть до последнего маркера, установленного опцией **Mark** (Метка), или все команды в буфере отмены, если маркер не был установлен)

Таблица 9.28 *(окончание)*

Событие	Описание
`:vlr-undoSubcommandNumber`	Команда UNDO (ОТМЕНИТЬ) выполняется с опцией, задающей количество отменяемых команд

Таблица 9.29. *Параметры, передаваемые в списке функции действия реактора типа* `:VLR-Undo-Reactor`

Событие	Кол-во пара-метров	Параметры
`:vlr-undoSubcommandAuto`	2	1 — целое число с кодом активности (всегда 4, т. к. данное событие происходит по окончании операции); 2 — символ, указывающий состояние режима **Auto** (Авто) (`T` — режим включен, `nil` — выключен)
`:vlr-undoSubcommandControl`	2	1 — целое число с кодом активности (всегда 4, т. к. данное событие происходит по окончании операции); 2 — целое число, указывающее состояние режима **Control** (Управление) (0 — выбрана опция **None** (Ничего), 1 — выбрана опция **One** (Одну), 2 — выбрана опция **All** (Все))
`:vlr-undoSubcommandBegin`, `:vlr-undoSubcommandEnd`, `:vlr-undoSubcommandMark`, `:vlr-undoSubcommandBack`	1	Всегда 0 (код активности — событие происходит до выполнения действий команды UNDO (ОТМЕНИТЬ))
`:vlr-undoSubcommandNumber`	2	1 — всегда 0 (код активности); 2 — целое число, указывающее количество отменяемых операций

9.5.17. Функция *vlr-wblock-reactor*

Создает реактор типа `:VLR-Wblock-Reactor` (реактор событий, связанных с командой WBLOCK (ПБЛОК)).

```
(vlr-wblock-reactor <данные> <реакции>)
```

Аргументы: `<данные>` — значение любого типа, применяемого в AutoLISP (`nil`, если данных нет); `<реакции>` — список из точечных пар с именами событий и функций действия.

Имена событий для реакторов этого типа должны выбираться из табл. 9.30.

Возвращаемое значение — VLR-объект (реактор).

Помимо аргумента с именем вызывающего реактора в данном случае функции действия передается аргумент — список, элементы которого перечислены в табл. 9.31.

Таблица 9.30. *События, используемые реакторами типа* `:VLR-Wblock-Reactor`

Событие	Описание
`:vlr-wblockNotice`	Начинается команда WBLOCK (ПБЛОК)
`:vlr-beginWblockPt`	Операция выполняется с набором объектов
`:vlr-beginWblockId`	Операция выполняется с блоком
`:vlr-beginWblock`	Операция выполняется со всем рисунком (уведомление о событии генерируется, только когда все объекты пространства модели исходного рисунка будут скопированы в другую базу)
`:vlr-otherWblock`[1]	Операция выполняется с базой рисунка (объекты скопированы, но их ID еще не пересчитаны). В этот момент можно клонировать дополнительные объекты (такие как словари и элементы словарей) — иначе они не будут скопированы — аналогично событию `beginDeepCloneXlation`
`:vlr-abortWblock`[2]	Операция была прервана до полного завершения
`:vlr-endWblock`	Операция завершена успешно
`:vlr-beginWblockObjects`	Началась операция пересчета ID объектов

Таблица 9.31. *Параметры, передаваемые в списке функции действия реактора типа* `:VLR-Wblock-Reactor`

Событие	Кол-во параметров	Параметры
`:vlr-wblockNotice`	1	VLA-объект исходной базы, содержащей копируемые объекты
`:vlr-beginWblockPt`	3	1 — VLA-объект целевой базы; 2 — VLA-объект исходной базы с копируемыми объектами; 3 — трехмерная точка, используемая в целевом рисунке как база копирования
`:vlr-beginWblockId`	3	1 — VLA-объект целевой базы; 2 — VLA-объект исходной базы; 3 — указатель входа в таблицу блоков для копируемого блока

[1] В официальной документации отсутствует.

[2] В официальной документации отсутствует.

Таблица 9.31 *(окончание)*

Событие	Кол-во пара-метров	Параметры
`:vlr-beginWblock:`, `vlr-otherWblock`	2	1 — VLA-объект целевой базы; 2 — VLA-объект исходной базы
`:vlr-abortWblock`, `:vlr-endWblock`	1	VLA-объект целевой базы
`:vlr-beginWblockObjects`	2	1 — VLA-объект базы рисунка, в который выполняется копирование; 2 — таблица соответствия ID

9.5.18. Функция *vlr-window-reactor*

Создает реактор типа `:VLR-Window-Reactor` (реактор событий, связанных с изменениями окон, с которыми работает система AutoCAD).

```
(vlr-window-reactor <данные> <реакции>)
```

Аргументы: `<данные>` — значение любого типа, применяемого в AutoLISP (`nil`, если данных нет); `<реакции>` — список из точечных пар с именами событий и функций действия.

Имена событий для реакторов этого типа должны выбираться из табл. 9.32.

Возвращаемое значение — VLR-объект (реактор).

Помимо аргумента с именем вызывающего реактора в данном случае функции действия передается аргумент — список, элементы которого перечислены в табл. 9.33.

Таблица 9.32. *События, используемые реакторами типа* `:VLR-Window-Reactor`

Событие	Описание
`:vlr-docFrameMovedOrResized`	Изменены границы окна с документом
`:vlr-mainFrameMovedOrResized`	Изменены границы окна системы AutoCAD

Таблица 9.33. *Параметры, передаваемые в списке функции действия реактора типа* `:VLR-Window-Reactor`

Событие	Кол-во пара-метров	Параметры
`:vlr-docFrameMovedOrResized`, `:vlr-mainFrameMovedOrResized`	2	1 — целое число с дескриптором (HWND) окна; 2 — символ (`T` — если окно было перемещено, `nil` — если изменились размеры окна)

9.5.19. Функция *vlr-xref-reactor*

Создает реактор типа `:VLR-XREF-Reactor` (реактор событий, связанных с операциями над внешними ссылками).

```
(vlr-xref-reactor <данные> <реакции>)
```

Аргументы: `<данные>` — значение любого типа, применяемого в AutoLISP (`nil`, если данных нет); `<реакции>` — список из точечных пар с именами событий и функций действия.

Имена событий для реакторов этого типа должны выбираться из табл. 9.34.

Возвращаемое значение — VLR-объект (реактор).

Помимо аргумента с именем вызывающего реактора в данном случае функции действия передается аргумент — список, элементы которого перечислены в табл. 9.35.

Таблица 9.34. *События, используемые реакторами типа* `:VLR-XREF-Reactor`

Событие	Описание
`:vlr-beginAttach`	Начало операции вставки внешней ссылки
`:vlr-otherAttach`	Объекты внешней ссылки уже добавлены, а ID объектов еще не транслированы. Данное событие следует сразу за событием `beginDeepCloneXlation`
`:vlr-abortAttach`	Операция вставки внешней ссылки прервана
`:vlr-endAttach`	Операция вставки внешней ссылки успешно завершена
`:vlr-redirected`	ID объекта из внешней ссылки заменяется на ID объекта в файле, на который сделана ссылка
`:vlr-comandeered`	ID объекта добавлен к таблице символов рисунка, в который вставлена ссылка
`:vlr-beginRestore`	Существующая внешняя ссылка будет разрешаться (при загрузке рисунка с внешними ссылками)
`:vlr-abortRestore`	Операция выгрузки или обновления внешней ссылки прервана до ее полного завершения
`:vlr-endRestore`	Существующая внешняя ссылка была разрешена (например, когда рисунок загрузился полностью)
`:vlr-xrefSubcommandBindItem`	Активизирована опция **Bind** (Внедрить) команды XREF (ССЫЛКА) или внедряется ранее существовавшая внешняя ссылка. Эта опция работает в интерактивном режиме и вызывает несколько событий
`:vlr-xrefSubcommandAttachItem`	Активизирована опция **Attach** (Вставить) команды XREF (ССЫЛКА) или разрешается ранее существовавшая внешняя ссылка. Эта опция работает в интерактивном режиме и вызывает несколько событий

Таблица 9.34 *(окончание)*

Событие	**Описание**
`:vlr-xrefSubcommandOverlayItem`	Активизирована опция **Overlay** (Наложить) команды XREF (ССЫЛКА) или разрешается ранее существовавшая внешняя ссылка. Эта опция работает в интерактивном режиме и вызывает несколько событий
`:vlr-xrefSubcommandDetachItem`	Активизирована опция **Detach** (Удалить) команды XREF (ССЫЛКА). Эта опция работает в интерактивном режиме и вызывает несколько событий
`:vlr-xrefSubcommandPathItem`	Активизирована опция **Path** (Путь) команды XREF (ССЫЛКА). Эта опция работает в интерактивном режиме и вызывает несколько событий
`:vlr-xrefSubcommandReloadItem`	Активизирована опция **Reload** (Обновить) команды XREF (ССЫЛКА) или обновляется ранее существовавшая внешняя ссылка. Эта опция работает в интерактивном режиме и вызывает несколько событий
`:vlr-xrefSubcommandUnloadItem`	Активизирована опция **Unload** (Выгрузить) команды XREF (ССЫЛКА) или выгружается ранее существовавшая внешняя ссылка

Таблица 9.35. *Параметры, передаваемые в списке функции действия реактора типа* `:VLR-XREF-Reactor`

Событие	**Кол-во параметров**	**Параметры**
`:vlr-beginAttach`	3	1 — VLA-объект (целевая база); 2 — строка с именем файла внешней ссылки; 3 — VLA-объект (база того рисунка, который вставляется как ссылка)
`:vlr-otherAttach`	2	1 — VLA-объект (целевая база рисунка); 2 — VLA-объект (исходная база рисунка, который вставляется как ссылка)
`:vlr-abortAttach`	1	VLA-объект (исходная база)
`:vlr-endAttach`	1	VLA-объект (целевая база)
`:vlr-redirected`	2	1 — целое число с ID объекта для перенаправления записи из таблицы символов из файла ссылки; 2 — целое число с ID объекта в целевом рисунке
`:vlr-comandeered`	3	1 — VLA-объект (целевая база); 2 — целое число с ID обрабатываемого объекта; 3 — VLA-объект (исходная база)

Таблица 9.35 *(продолжение)*

Событие	Кол-во пара-метров	Параметры
`:vlr-beginRestore`	3	1 — VLA-объект (целевая база); 2 — строка с именем записи в таблице блоков и ссылок; 3 — VLA-объект (исходная база)
`:vlr-abortRestore`, `:vlr-endRestore`	1	VLA-объект (целевая база)
`:vlr-xrefSubcommandBindItem`	2	1 — целое число, как показатель хода выполнения подкоманды (опции) **Bind** (Внедрить) (0 — вызвана подкоманда **Bind** (Внедрить); 2 — ссылка внедряется; 3 — ссылка внедрена; 4 — подкоманда завершена; 5 — подкоманда будет прервана; 6 — подкоманда завершена аварийно; 7 — передается для блока XDep, внедряемого командой XBIND (ВНЕДРИТЬ); 8 — передается для всех других символов: слоев, типов линий, текстовых стилей, размерных стилей); 2 — целое число с ID объекта внешней ссылки или 0
`:vlr-xrefSubcommandAttachItem`	2	1 — целое число, как показатель хода выполнения подкоманды (опции) **Attach** (Вставить) (0 — вызвана подкоманда; 2 — ссылка обрабатывается; 3 — ссылка обработана; 4 — подкоманда завершена; 5 — подкоманда будет прервана; 6 — подкоманда завершена аварийно); 2 — строка с именем вставляемого файла или `nil`
`:vlr-xrefSubcommandOverlayItem`	2	1 — целое число, как показатель хода выполнения подкоманды (опции) **Overlay** (Наложить) (0 — вызвана подкоманда; 2 — ссылка обрабатывается; 3 — ссылка обработана; 4 — подкоманда завершена; 5 — подкоманда будет прервана; 6 — подкоманда завершена аварийно); 2 — строка с именем вставляемого файла или `nil`
`:vlr-xrefSubcommandDetachItem`	2	1 — целое число, как показатель хода выполнения подкоманды (опции) **Detach** (Удалить) (0 — вызвана подкоманда; 2 — ссылка обрабатывается; 3 — ссылка обработана; 4 — подкоманда завершена; 5 — подкоманда будет прервана; 6 — подкоманда завершена аварийно); 2 — целое число с ID объекта внешней ссылки или 0

Таблица 9.35 *(окончание)*

Событие	Кол-во пара-метров	Параметры
`:vlr-xrefSubcommandPathItem`	3	1 — целое число, как показатель хода выполнения подкоманды (опции) **Path** (Путь) (0 — вызвана подкоманда; 2 — ссылка обрабатывается; 3 — ссылка обработана; 4 — подкоманда завершена; 5 — подкоманда будет прервана; 6 — подкоманда завершена аварийно); 2 — целое число с ID объекта внешней ссылки или 0; 3 — строка с новым путем внешней ссылки или `nil`
`:vlr-xrefSubcommandReloadItem`	2	1 — целое число, как показатель хода выполнения подкоманды (опции) **Reload** (Обновить) (0 — вызвана подкоманда; 2 — ссылка обрабатывается; 3 — ссылка обработана; 4 — подкоманда завершена; 5 — подкоманда будет прервана; 6 — подкоманда завершена аварийно); 2 — целое число с ID объекта внешней ссылки или 0
`:vlr-xrefSubcommandUnloadItem`	2	1 — целое число, как показатель хода выполнения подкоманды (опции) **Unload** (Выгрузить) (0 — вызвана подкоманда; 2 — ссылка обрабатывается; 3 — ссылка обработана; 4 — подкоманда завершена; 5 — подкоманда будет прервана; 6 — подкоманда завершена аварийно); 2 — целое число с ID объекта внешней ссылки или 0

9.5.20. Функция *vlr-reaction-set*

Добавляет или заменяет функцию действия в реакторе.

```
(vlr-reaction-set <реактор> <событие> '<функция>)
```

Аргументы: `<реактор>` — VLR-объект; `<событие>` — символ из числа зарезервированных для обозначения событий, обрабатываемых реакторами данного типа; `<функция>` — символ (с апострофом) с именем функции.

Возвращаемого значения нет.

Пример:

- `(vlr-reaction-set vr12 :vlr-commandWillStart 'vlr-trace-reaction)` заменяет в реакторе `vr12` старую функцию, действовавшую при событии `:vlr-commandWillStart`, на `vlr-trace-reaction`

9.6. Операции с реакторами

Расширение языка AutoLISP позволяет выполнять различные операции над реакторами. Можно получать списки реакторов, запрашивать данные, владельцев, имена функций действия. Допускается удаление, распространение, выключение, включение, модификация реакторов.

Осуществляющие все эти операции функции рассматриваются в *разд. 9.6.1—9.9.20*.

9.6.1. Функция *vlr-reactions*

Выводит список из точечных пар событий и функций действий.

```
(vlr-reactions <реактор>)
```

Аргумент `<реактор>` — VLR-объект.

Возвращаемое значение — список из точечных пар `(<событие> . <функция>)`.

9.6.2. Функция *vlr-remove*

Отключает реактор, не удаляя его из рисунка (такой реактор можно в дальнейшем активизировать).

```
(vlr-remove <реактор>)
```

Аргумент `<реактор>` — VLR-объект.

Возвращается значение аргумента `<реактор>`. В случае сбоя возвращается `nil`.

9.6.3. Функция *vlr-remove-all*

Отключает реакторы всех типов или реакторы определенного типа.

```
(vlr-remove-all [<тип>])
```

Аргумент `<тип>` может принимать значение одного из символов, зарезервированных для обозначений типов реакторов (см. табл. 9.2 и 9.3). Если аргумент опущен, то отключаются все реакторы.

Возвращаемое значение — список из списков с реакторами. Внутренние списки имеют вид: `(<тип> <VLR-объект> ...)`, где первый элемент — это зарезервированный символ для обозначения типа (например, `:VLR-Command-Reactor`), а остальные

элементы — VLR-объекты (реакторы). Если функция не отключила ни одного реактора, то возвращается `nil`.

9.6.4. Функция *vlr-add*

Включает (активизирует) отключенный реактор.

```
(vlr-add <реактор>)
```

Аргумент `<реактор>` — VLR-объект.

Возвращается значение аргумента `<реактор>`.

9.6.5. Функция *vlr-added-p*

Проверяет, включен ли реактор.

```
(vlr-added-p <реактор>)
```

Аргумент `<реактор>` — VLR-объект.

Возвращаемое значение — `T` (если реактор включен), `nil` (если реактор выключен).

9.6.6. Функция *vlr-current-reaction-name*

Определяет имя события, вызвавшего функцию действия реактора.

```
(vlr-current-reaction-name)
```

Возвращает зарезервированный символ с именем события, вызвавшего реактор.

9.6.7. Функция *vlr-data*

Извлекает данные, связанные с реактором (о данных, передаваемых функции действия, *см. разд. 9.5*).

```
(vlr-data <реактор>)
```

Аргумент `<реактор>` — VLR-объект.

Возвращаемое значение — данные, сохраненные вместе с реактором.

9.6.8. Функция *vlr-data-set*

Заменяет ранее записанные данные, связанные с реактором.

```
(vlr-data-set <реактор> <данные>)
```

Аргументы: `<реактор>` — VLR-объект; `<данные>` — значение любого типа, которое нужно сохранить с реактором вместо старых данных.

Возвращаемое значение — значение аргумента `<данные>`.

9.6.9. Функция *vlr-owner-add*

Добавляет объект к списку владельцев реактора типа `:VLR-Object-Reactor` (о владельцах реактора *см. разд. 9.5* и *9.5.4*).

```
(vlr-owner-add <реактор> <владелец>)
```

Аргументы: `<реактор>` — VLR-объект (реактор); `<владелец>` — VLA-объект, вносимый в список владельцев (объектов, которые могут активизировать функцию действия реактора).

Возвращаемое значение — значение аргумента `<владелец>`.

9.6.10. Функция *vlr-owner-remove*

Удаляет объект из списка владельцев реактора типа `:VLR-Object-Reactor`.

```
(vlr-owner-remove <реактор> <владелец>)
```

Аргументы: `<реактор>` — VLR-объект (реактор); `<владелец>` — VLA-объект, который удаляется из списка владельцев.

Возвращаемое значение — значение аргумента `<владелец>`.

9.6.11. Функция *vlr-owners*

Получает список владельцев реактора типа `:VLR-Object-Reactor`.

```
(vlr-owners <реактор>)
```

Аргумент `<реактор>` — VLR-объект (реактор объектов).

Возвращаемое значение — список, элементами которого являются VLA-объекты.

9.6.12. Функция *vlr-notification*

Выводит информацию об активности реактора в зависимости от пространства имен.

```
(vlr-notification <реактор>)
```

Аргумент `<реактор>` — VLR-объект.

Возвращаемое значение — один из двух зарезервированных символов:

- `all-documents` — реактор работает во всех документах;
- `active-document-only` — реактор работает только в активном документе.

9.6.13. Функция *vlr-set-notification*

Задает режим работы реактора, в зависимости от активности нужного документа.

```
(vlr-set-notification <реактор> '<документы>)
```

Аргументы: `<реактор>` — VLR-объект (реактор); `<документы>` — один из двух зарезервированных символов (`all-documents` — реактор работает во всех документах, `active-document-only` — реактор работает только в активном документе). Перед вторым аргументом должен присутствовать апостроф.

Возвращаемое значение — значение аргумента `<реактор>`.

9.7. Временные и постоянные реакторы

По умолчанию создаваемые реакторы являются *временными* (transient), т. е. не сохраняются в рисунке после его закрытия. Однако в технологии ActiveX системы AutoCAD предусмотрена операция смены режима работы реакторов с временного на постоянный.

Постоянные (persistent) реакторы сохраняются прямо в рисунке и возобновляют свое функционирование после открытия рисунка в следующем сеансе работы системы AutoCAD (при условии загрузки тех функций, которые фигурируют в реакторах в качестве функций действия). Перевод реактора в постоянный режим может быть осуществлен с помощью функции `vlr-pers`, возврат во временный — с помощью функции `vlr-pers-release`.

Для удобства пользователей добавлены также функция `vlr-pers-p`, проверяющая режим работы реактора на постоянность, и `vlr-pers-list`, которая выводит список постоянных реакторов.

Перечисленные функции рассматриваются в *разд. 9.7.1—9.7.4*.

9.7.1. Функция *vlr-pers*

Переводит реактор в постоянный режим.

```
(vlr-pers <реактор>)
```

Аргумент `<реактор>` — VLR-объект (реактор).

При нормальном завершении функция `vlr-pers` возвращает значение аргумента `<реактор>`. Если операция перевода не выполнилась, то функция возвращает `nil`.

9.7.2. Функция *vlr-pers-release*

Переводит реактор во временный режим.

```
(vlr-pers-release <реактор>)
```

Аргумент `<реактор>` — VLR-объект.

Функция `vlr-pers-release` в случае успешного завершения возвращает значение аргумента `<реактор>`. В противном случае возвращается `nil`.

9.7.3. Функция *vlr-pers-p*

Проверяет реактор на постоянность.

```
(vlr-pers-p <реактор>)
```

Аргумент <реактор> — VLR-объект (реактор).

Данная функция возвращает значение аргумента <реактор>, если проверяемый реактор является постоянным. Если реактор является временным, то возвращается nil.

9.7.4. Функция *vlr-pers-list*

Получает список постоянных реакторов в текущем документе.

```
(vlr-pers-list [<тип>])
```

Аргумент <тип> — зарезервированный символ, обозначающий тип реактора (см. табл. 9.2 и 9.3).

Возвращаемое значение — список с VLR-объектами (постоянными реакторами указанного типа). Если аргумент <тип> не передан, то возвращается список постоянных реакторов всех типов.

9.8. Правила работы с реакторами

Реакторы являются мощными элементами управления, создаваемыми с помощью современной технологии ActiveX. Но такой сильный аппарат должен использоваться достаточно аккуратно, ибо неправильное его применение может привести к нежелательным результатам, вплоть до подвешивания операционной системы или досрочному прекращению работы AutoCAD.

При проектировании реакторов фирма Autodesk рекомендует учитывать следующие соображения.

1. Естественная последовательность событий иногда может нарушаться. Например, команда OPEN (ОТКРЫТЬ) вызывает события :vlr-commandWillStart, :vlr-beginDwgOpen, :vlr-endDwgOpen, :vlr-commandEnded. Однако эти четыре события не обязательно произойдут именно в указанном порядке. Достоверно можно утверждать только, что событие :vlr-commandWillStart предшествует событию :vlr-commandEnded, а событие :vlr-beginDwgOpen — событию :vlr-endDwgOpen.
2. Последовательность работы функций действия реакторов не всегда соответствует последовательности событий, вызвавших работу этих реакторов.
3. В функциях действия нельзя использовать такие интерактивные функции, как getkword, getpoint, entsel и т. д. Кроме того, запрещается в функциях действия применять функцию command.

4. Нельзя вызывать диалоговые окна во время обработки событий. Функции, работающие с диалоговыми окнами, относятся к интерактивным функциям и поэтому будут конфликтовать с работой реакторов.
5. Объект, который стал источником события, не может быть модифицирован из функции действия, реагирующей на это событие.
6. Нельзя из функции действия, обрабатывающей событие, выполнять операции, которые могли бы привести к отмене данного события (это может вызвать зацикливание).
7. Перед созданием реактора нужно проверять, не создан ли такой реактор ранее. Иначе может происходить многократная обработка одного и того же события.
8. Во время диалоговых запросов системы AutoCAD никакие события не происходят.

9.9. Примеры реакторов

В данном разделе будут приведены несложные примеры реакторов. Дополнительные примеры функций с реакторами можно найти в папке Sample\VisualLISP \reactors, расположенной внутри папки с программным обеспечением системы AutoCAD.

9.9.1. Схема создания реакторов

Вспомним схему создания реакторов, приведенную в *разд. 9.5*. Необходимо выполнить выражение следующего вида:

```
(vlr-<тип>-reactor <данные> <реакции>)
```

Вместо `<тип>` следует подставить тип реактора из табл. 9.2 и 9.3 (`acdb`, `docmanager`, `command`, `deepclone` и т. д.). У реактора объектов (когда `<тип>` равен `object`) перед аргументом `<данные>` присутствует еще один аргумент — `<владельцы>`, по которому определяется, какие объекты вызвали реактор.

Аргумент `<данные>` — это LISP-данное любого вида (число, список, строка, `nil` и т. п.), которое при срабатывании реактора передается функции действия (она может получить это данное из реактора с помощью специальной функции `vlr-data`).

Аргумент `<реакции>` — это список, в котором задаются события, отслеживаемые реактором, и функции, вызываемые в качестве реакции при получении уведомления о своем событии. Список должен иметь такой вид:

```
(list '(<событие1> . <функция1>) '(<событие2> . <функция2>) ...)
```

Здесь для каждого из отслеживаемых событий (`<событие1>`, `<событие2>` и т. д.) создается точечная пара, вторым элементом которой является соответствующая

функция действия (<функция1>, <функция2> и т. д.). Пример (для реактора базы рисунка):

```
(list '(:vlr-objectAppended . funA) '(:vlr-objectModified . funM))
```

Функции действия, которые являются вторым элементом в приведенных точечных парах, должны иметь два аргумента (исключением являются функции действия реактора объектов, которые должны иметь три аргумента):

- VLR-объект (реактор);
- список параметров (его формирует AutoCAD).

У функции действия реактора объектов должен быть еще дополнительный первый аргумент, содержащий VLA-объект владельца реактора (т. е. объекта, который вызвал реактор).

9.9.2. Реактор изменений базы рисунка

В первом примере создадим реактор, который отслеживает любые изменения базы рисунка. Используем в качестве функции действия стандартную функцию `vlr-beep-reaction`. Текст функции `reac1`, создающей реактор, приведен в листинге 9.1.

Листинг 9.1. Функция `reac1`

```
; Глава 09\Book01\reac1.lsp
(defun reac1 ( / )
; Создание реактора базы (для трех событий)
(setq dbr (vlr-acdb-reactor nil
  (list '(:vlr-objectAppended . vlr-beep-reaction)
        '(:vlr-objectErased . vlr-beep-reaction)))
);setq
);defun reac1
```

С помощью функции `reac1` в рисунке создается объект `dbr`, являющийся реактором типа `:VLR-AcDb-Reactor`, т. е. реактор изменений базы рисунка. Этот реактор отслеживает только два события: `:vlr-objectAppended` и `:vlr-objectErased` — добавления и удаления объектов. Он не будет реагировать на модификацию объектов или их восстановление после удаления.

Поскольку функцией действия выбрана функция `vlr-beep-reaction`, то при построении или удалении примитивов раздается звуковой сигнал. В версии 2006 звуковой сигнал почему-то раздается и при прохождении курсора над выделенной ручкой объекта (специфика внутренней реализации этой версии?).

Для отключения реактора можно использовать функции `vlr-remove` и `vlr-remove-all`.

9.9.3. Реактор DXF-операций

Следующий пример связан с реактором типа `:VLR-DXF-Reactor`. Такой реактор должен следить за операциями загрузки DXF-файла или сохранения рисунка в формате DXF. В листинге 9.2 показан текст функции `reac2`, создающей реактор.

Листинг 9.2. Функция `reac2`

```
; Глава 09\Book02\reac2.lsp
(defun reac2 ( / )
; Создание DXF-реактора (для четырех событий)
(setq dxfr (vlr-dxf-reactor nil
  (list '(:vlr-abortDxfIn . vlr-beep-reaction)
        '(:vlr-dxfInComplete . good1)
        '(:vlr-dxfOutComplete . good2)
        '(:vlr-abortDxfOut . vlr-beep-reaction)))
);setq
);defun reac2
```

Функция `reac2` создает реактор DXF-операций, который отслеживает четыре события. Для двух событий (`:vlr-abortDxfIn` и `:vlr-abortDxfOut`) реактор с помощью функции `vlr-beep-reaction` выдает краткий звуковой сигнал (его можно считать оповещением об ошибке), а еще для двух событий, которые можно считать признаками успешного окончания операции, выполняет функцию `good1` или функцию `good2`. Текст этих функций приведен в листинге 9.3.

Листинг 9.3. Функции действия для DXF-реактора

```
; Глава 09\Book02\cback2.lsp
(defun good1 ( / )
(alert "Импорт из DXF-файла успешно завершен")
); defun good1
(defun good2 ( / )
(princ "\nСохранение в DXF-файл выполнено ")
); defun good2
```

Первая из приведенных в листинге функций должна вызываться по окончании импорта из DXF-файла и выводить окно функции `alert` с фиксированным текстом, а вторая функция действия тоже должна выводить сообщение, но в командную строку.

Обе эти функции имеют два аргумента, что определяется жесткостью структуры функций действия. Первый аргумент — это реактор (VLR-объект), а второй — данные, формируемые системой AutoCAD. Как следует из *разд. 9.5.9*, этот аргумент равен `nil`, однако по синтаксису не может быть опущен.

Попробуйте выполнить экспорт в DXF-файл и получите в командной строке сообщение ***Сохранение в DXF-файл выполнено***.

Однако если вы выполните импорт DXF-файла (например, с помощью команды INSERT (ВСТАВИТЬ)), то получите окно функции `alert` с сообщением не один, а три раза (причем два раза до завершения операции). Это подтверждает справедливость предупреждения, сделанного в *разд. 9.8*: нельзя полагаться на естественную последовательность генерации событий. Реакторы — коварная вещь.

9.9.4. Реактор команды ERASE

Реактор команд позволяет контролировать события выполнения определенных команд. В листинге 9.4 приведена программа, которая с помощью реактора препятствует удалению примитивов по команде ERASE (СТЕРЕТЬ).

Листинг 9.4. Функция `reac3`

```
; Глава 09\Book03\reac3.lsp
(defun reac3 ( / )
; Создание реактора команд
(setq cmr (vlr-command-reactor "Реактор команд: "
  (list '(:vlr-commandEnded . undel)))
);setq
);defun reac3
```

Функция действия `undel`, которая срабатывает по событию `:vlr-commandEnded`, показана в листинге 9.5.

Листинг 9.5. Функция действия реактора команд

```
; Глава 09\Book03\cback3.lsp
(defun undel (reac cname / _ad)
(princ (strcat "\n" (vlr-data reac) " контролирую "))
(princ cname)
(setq _ad (vla-get-ActiveDocument (vlax-get-acad-object)))
; Проверка имени команды
(if (member (getcname (car cname)) (list "ERASE" "_ERASE"))
  (progn
    (vla-SendCommand _ad "_.UNDO ")
    (vla-SendCommand _ad "1 ")
  );progn
);if
);defun undel
```

Формирование реактора в функции `reac3` выполняется с помощью выражения `(vlr-command-reactor "Реактор команд: " (list '(:vlr-commandEnded . undel)))`. Здесь, согласно схеме создания реакторов, `<данные>` — это строка, а `<реакции>` — это список `(list '(:vlr-commandEnded . undel))`. В списке приведено только одно отслеживаемое событие `:vlr-commandEnded`, которое генерируется по окончании команды.

Функция `undel` имеет два аргумента: `reac` и `cname`. Первый из них — это VLR-объект реактора (т. е. значение глобальной переменной `cmr` из функции `reac3`). Второй — список с именем команды, которую только что выполнила система AutoCAD.

Функция действия с помощью выражения `(vlr-data reac)` извлекает данные, записанные в реактор (в нашем примере — это строка `"Реактор команд: "`), которые затем используются функцией `princ` при печати.

Когда в рисунке происходит событие `:vlr-commandEnded`, соответствующее окончанию команды AutoCAD, то функция действия извлекает имя этой команды из списка `cname` и проверяет, является ли команда той, которую отслеживает реактор (в нашем примере отслеживаются два варианта одной команды ERASE (СТЕРЕТЬ), без подчеркивания и с подчеркиванием). Если встретилась именно эта команда, то тут же выполняется команда UNDO (ОТМЕНИТЬ) с опцией **1**, которая отменяет последнее действие. Поэтому удаленные объекты тут же восстанавливаются.

При создании реактора следует учитывать, что существует такой режим команды UNDO (ОТМЕНИТЬ), при котором откаты невозможны. Этот режим необходимо обрабатывать специальным образом (развитие реактора предоставляется читателю).

Замечание

Обратите внимание на то, что для вызова команды в функции действия используется функция `vla-SendCommand`, соответствующая методу `SendCommand`, а не запрещенная в реакторах функция `command`.

9.9.5. Отслеживание двойного щелчка (реактор мыши)

Реактор мыши дает возможность контролировать события мыши (одиночные или двойные щелчки). В листингах 9.6 и 9.7 приведены функции, которые в составе реактора отслеживают двойной щелчок на примитиве и изменяют цвет примитива, увеличивая номер цвета на 1. Используется свойство `Color`, а не `TrueColor`. Для правильной работы реактора необходимо отключить обработку двойного щелчка самой системой AutoCAD (см. команду DBLCLKEDIT (ДЩРЕДАКТ)).

Листинг 9.6. Функция reac4

```
; Глава 09\Book04\reac4.lsp
(defun reac4 ( / )
; Создание реактора мыши (на двойной щелчок)
(setq mouser (vlr-mouse-reactor "Реактор мыши: "
  (list '(:vlr-beginDoubleClick . ccol)))
);setq
);defun reac4
```

Листинг 9.7. Функция действия реактора мыши

```
; Глава 09\Book04\cback4.lsp
(defun ccol (reac argpoint / _ss1 _ve1 _col)
(princ (strcat "\n" (vlr-data reac) "двойной щелчок. Точка "))
(princ (car argpoint))
(princ " ")
(setq _ss1 (ssget (car argpoint)))
; По двойному щелчку реактор увеличивает цвет объекта на 1 (не более 256)
; и перемещает объект на 100 мм вправо
(if _ss1
  (progn
    (setq _ve1 (vlax-ename->vla-object (ssname _ss1 0)))
    (setq _col (1+ (vla-get-Color _ve1)))
    (if (> _col 256)(setq _col 1))
    ; Изменение цвета
    (vla-put-Color _ve1 _col)
    ; Сдвиг объекта
    (vla-Move _ve1 (vlax-3D-point 0.0 0.0 0.0)
      (vlax-3D-point 100.0 0.0 0.0))
  );progn
);if
);defun ccol
```

В этом примере функция `reac4` создает реактор на событие двойного щелчка мыши. Если щелчок выполнен на примитиве, то у примитива с помощью функции `vla-put-Color` цвет увеличивается на 1 (кроме случая цвета 256 — следующим становится цвет с номером 1), и объект с помощью функции `vla-Move` сдвигается на 100 мм вправо. Кроме того, реактор выводит кординаты точки щелчка. Если щелчок выполнен вне объектов, то выводится только текст с координатами.

9.9.6. Реактор отрезков

Этот пример уже ранее публиковался. Создадим реактор, который является реактором типа `:VLR-Object-Reactor` для отрезков и при модификации этих отрезков выводит их новую длину. Обрабатывается событие `:vlr-modified`.

Назовем функцию действия, которая должна обрабатывать выбранное событие, `showlen`. Как функция действия реактора объектов, она должна получать не два аргумента, как в предыдущих примерах, а три:

- VLA-объект (один из владельцев реактора);
- VLR-объект (реактор);
- список параметров (для рассматриваемого события он пуст, согласно табл. 9.9).

В листинге 9.8 приведен текст функции `reac5`, создающей реактор объектов и формирующей список его владельцев. Функция действия для этого реактора (`showlen`) показана в листинге 9.9.

Листинг 9.8. Функция `reac5`

```
; Глава 09\Book05\reac5.lsp
(defun reac5 ( / ww i wl ae)
  (princ "\nВыберите отрезки: ")
  (setq ww (ssget))
  (if ww
    (progn
      (princ (strcat "\nВыбрано " (itoa (sslength ww)) " объектов. "))
      (vl-load-com)
      (setq axl nil)
     ; Формирование списка владельцев реактора
      (setq i -1 wl (sslength ww))
      (repeat wl
        ; Преобразование элементов набора ww в VLA-объекты
        (setq i (1+ i) ae (vlax-ename->vla-object (ssname ww i)))
        (setq axl (append axl (list ae)))
      ); repeat
     ; Создание реактора
      (setq liner
        (vlr-object-reactor axl "Реактор отрезков; новая длина "
                            (list '(:vlr-modified . showlen)))
      );setq
    );progn
  );if
);defun reac5
```

Листинг 9.9. Функция действия реактора отрезков

```
; Глава 09\Book05\cback5.lsp
(defun showlen (vlao reac args /)
  (princ (strcat "\n" (vlr-data reac) (rtos (vla-get-Length vlao) 2 2)))
);defun showlen
```

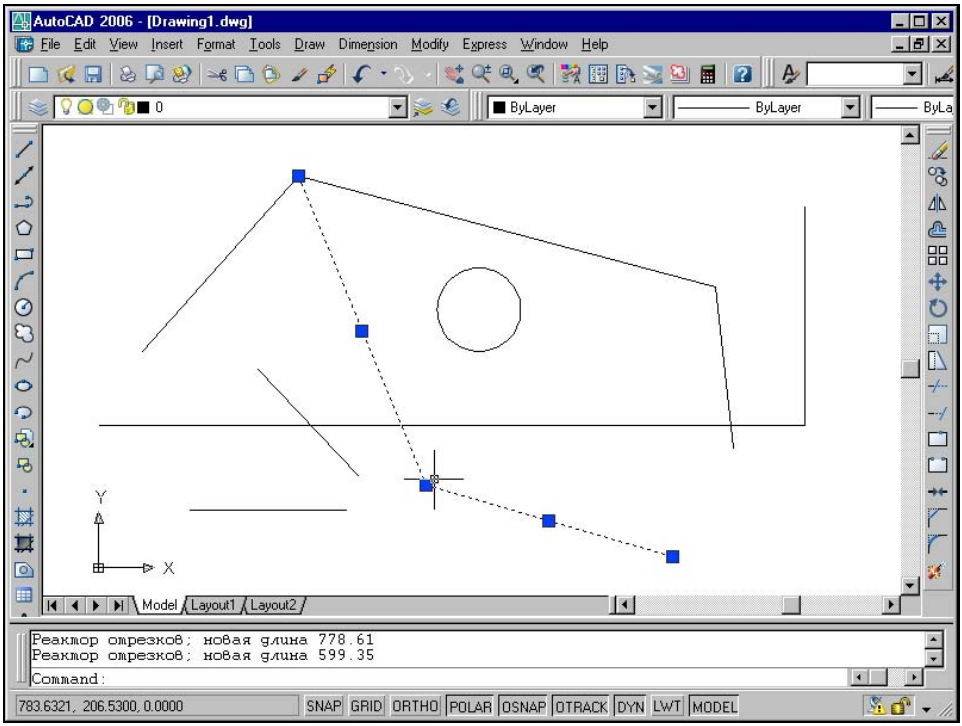


Рис. 9.1. Пример работы реактора объектов

На рис. 9.1 построено девять отрезков (есть еще одна окружность, но она не важна для нас). Два выделенных отрезка этого документа имеют общую точку.

После выполнения функции `reac5` в рисунке образовался набор из выбранных пользователем отрезков, которым был назначен реактор типа `:VLR-Object-Reactor` (он сохранен в переменной `liner`).

В рисунке осуществлена операция одновременного редактирования двух подсвеченных отрезков (сначала были выделены объекты, а затем пользователь выбрал ручку, расположенную в общей точке двух отрезков и переместил ее). В результате такого перемещения ручек с двумя объектами произошло событие `:vlr-modified`, а наблюдавший за этим реактор `liner` выдал сообщения:

```
Реактор отрезков; новая длина 778.61
Реактор отрезков; новая длина 599.35
```

Для вычисления новой длины отрезков была использована функция `vla-get-Length`, которая извлекает из VLA-объекта его текущую длину.

В случае необходимости сохранения реактора вместе с документом в функцию `reac5` следует добавить строку:

```
(vlr-pers liner)
```

Но даже после сохранения реактора вместе с рисунком нужно позаботиться о загрузке в следующем сеансе AutoCAD функции `showlen`, без которой реактор будет выдавать сообщение об ошибке.

9.9.7. Связанные окружности (реактор объектов)

В следующем примере демонстрируются реакторы объектов, которые работают с двумя незримо связанными окружностями. Над этими окружностями можно выполнять операции изменения с помощью ручек или команд редактирования с описанными ниже результатами.

При перемещении первой окружности вторая окружность сама переносится внутрь первой с совмещением центров, получает радиус на 10 мм меньше, чем у первой, и увеличивает свой номер цвета на 1. Если изменять радиус одной из этих окружностей, то изменяется радиус и другой окружности, но таким образом, чтобы радиусы, если они достаточно велики, отличались на 10 мм. Выдерживаются ограничения: минимальное значение первого радиуса — 20 мм, второго — 10 мм. Первая окружность своего цвета не меняет.

Программный код функции создания реакторов показан в листинге 9.10, а функций действия — в листинге 9.11.

Листинг 9.10. Функция `reac6`

```
; Глава 09\Book06\reac6.lsp
(defun reac6 ( / _wc1 _wc2 _ac1 _ac2)
(vl-load-com)
(princ "\nВыберите 1-ю окружность: ")
(while (not _wc1)
  (setq _wc1 (car (entsel)))
  (if (/= "CIRCLE" (cdr (assoc 0 (entget _wc1))))(setq _wc1 nil))
);while
(princ "\nВыберите 2-ю окружность: ")
(while (not _wc2)
  (setq _wc2 (car (entsel)))
  (if (/= "CIRCLE" (cdr (assoc 0 (entget _wc2))))(setq _wc2 nil))
) ;while
; Преобразование выбранных примитивов в VLA-объекты
(setq _ac1 (vlax-ename->vla-object _wc1))
(setq _ac2 (vlax-ename->vla-object _wc2))
; Создание двух реакторов окружностей
(setq creactor1 (vlr-object-reactor (list _ac1) (list _ac2)
  (list '(:vlr-modified . rchc1))))
(setq creactor2 (vlr-object-reactor (list _ac2) (list _ac1)
  (list '(:vlr-modified . rchc2))))
);defun reac6
```

Листинг 9.11. Функции действия реакторов окружностей

```
; Глава 09\Book06\cback6.lsp
(defun rchc1 (vlao reac args / )
; Убираем с помощью vl-catch-all-apply сообщение:
;; error: Automation Error. Object was notifying
(vl-catch-all-apply 'chc1 (list vlao reac args))
(princ)
);defun rchc1
(defun rchc2 (vlao reac args / )
(vl-catch-all-apply 'chc2 (list vlao reac args))
(princ)
);defun rchc2
; Вспомогательные функции
(defun chc1 (vlao reac args / _ac2 r1 r2 pc1)
; Читаем данные 1-й окружности
(setq r1 (vla-get-Radius vlao))
(setq pc1 (vla-get-Center vlao))
; Меняем 2-ю окружность
(setq _ac2 (car (vlr-data reac)))
(if (> r1 10.0)
  (progn
    (setq r2 (max 10.0 (- r1 10.0)))
    (vla-put-Radius _ac2 r2)
  );progn
);if
(vla-put-Center _ac2 pc1)
(vla-put-Color _ac2 (rem (1+ (vla-get-Color _ac2)) 256))
);defun chc1
(defun chc2 (vlao reac args / _ac1 r1 r2 pc2)
; Читаем данные 2-й окружности
(setq r2 (vla-get-Radius vlao))
(setq pc2 (vla-get-Center vlao))
; Меняем 1-ю окружность
(setq _ac1 (car (vlr-data reac)))
(setq r1 (max 20.0 (+ r2 10.0)))
(vla-put-Radius _ac1 r1)
(vla-put-Center _ac1 pc2)
);defun chc2
```

В приведенных листингах пять функций. Функция `reac6` создает два объектных реактора, владельцами которых являются две указываемые в интерактивном режиме окружности. Реактор `creactor1`, владельцем которого является первая окружность, хранит также в себе данные (VLA-объект) о второй окружности. Ана-

логично второй реактор `creactor2` (его владельцем является вторая окружность) хранит VLA-объект первой окружности.

В качестве функций действия выступают функции `rchc1` и `rchc2`. Когда два объекта связаны реакторами, то может возникнуть бесконечный вызов реакторов одного за другим. Система AutoCAD это проверяет, прерывает связь при повторном вызове реактора и выдает сообщение:

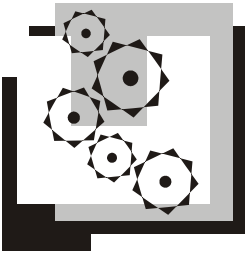
; error: Automation Error. Object was notifying

(; ошибка: Ошибка Automation. Объект посылал уведомление)

Функция `rchc1` подавляет это ненужное в нашей ситуации сообщение с помощью функции `vl-catch-all-apply` и передает управление функции `chc1`, которая корректирует второй примитив. Аналогично организована работа функций `rchc2` и `chc2`.

Изменение свойств окружностей выполняется с помощью функций `vla-put-Color`, `vla-put-Radius` и `vla-put-Center`.

ГЛАВА 10



Приемы программирования

В этой главе мы рассмотрим некоторые практические приемы программирования, которым не уделено в литературе достаточно внимания. Иллюстрация этих приемов будет проводиться на функциях, разработанных авторами книги для практического применения, а также на упрощенных функциях, специально подготовленных для данного издания. Ввиду ограниченности количества оптимальных решений простых задач программирования в AutoCAD некоторые функции могут быть очень похожи на решения других LISP-программистов или полностью совпадать с ними. Авторы не имеют возможности сверки своих решений со всеми другими программными решениями в этой сфере на предмет эксклюзивности. Все функции, приведенные в данной главе, являются оригинальными разработками авторов без использования каких-либо внешних источников. Комментарии к приведенным решениям носят информационный характер, без глубокого изложения теоретических основ и предназначены для пояснения алгоритмов работы этих функций.

10.1. Рекурсия

Некоторые принципиальные моменты, связанные с рекурсией (обращением функции к самой себе), уже рассматривались авторами в рамках другой книги[1]. Здесь мы попробуем показать некоторые примеры реального применения рекурсивных функций и прокомментировать положительные и отрицательные результаты использования рекурсии в практическом программировании. Напомним только некоторые выводы, сделанные по результатам упомянутого материала.

Первый вывод — это обязательность компиляции программ на AutoLISP (в том числе с применением расширений Visual LISP), если в программе используются

---

[1] Зуев С. А., Полещук Н. Н. САПР на базе AutoCAD — как это делается. — СПб.: БХВ-Петербург, 2004. — 1168 с. (глава 10).

рекурсивные функции. Компиляция положительно сказывается и на программах, в которых нет рекурсивных функций, так что эта рекомендация может рассматриваться как универсальная, без привязки к каким-либо приемам программирования.

Второй вывод заключается в том, что наиболее эффективным применение рекурсии является в случае необходимости обработки неоднородных списков неизвестной или переменной вложенности, когда использование других способов затруднено или связано с существенно большим объемом кода.

Третий вывод такой, что применение рекурсии не является панацеей и его применение должно разумно сочетаться с использованием встроенных функций и итерации.

Кроме того, для рекурсивных функций можно считать характерными следующие свойства: компактность и трудность восприятия. Компактность обусловлена тем, что рекурсивные функции обычно имеют меньшее количество аргументов по сравнению с другими алгоритмами. Трудность восприятия возникает вследствие того, что рекурсивный вызов, как правило, имеет неочевидный возврат.

10.1.1. Функция *PL:String-Rep*

Описание: заменяет в строке все вхождения старой подстроки на новую. Чувствительна к регистру символов.

Аргументы:

- <исходная строка>, <искомая подстрока>, <новая подстрока> — строки.

Примечание

Передача аргументов другого типа приводит к ошибке.

Возврат: строка. Текст функции приведен в листинге 10.1.

Листинг 10.1. Функция `PL:String-Rep`

```
; Глава 10\Book01\PL_String-Rep.lsp
(defun PL:String-Rep (_str _old _new / _pos)
  (if (setq _pos (vl-string-search _old _str))
;;; Если в строке обнаружено вхождение искомой подстроки, то запоминается
;;; его начальная позиция в переменной _pos.
    (strcat (substr _str 1 _pos) _new
;;; Отрезается фрагмент исходной строки, от ее начала и до начала
;;; искомой строки. Функция strcat ожидает возврата следующего
;;; рекурсивного вызова. По завершению этого вызова фрагмент исходной
;;; строки, новая подстрока и возврат рекурсивного вызова сцепляются
;;; в одну строку.
```

```
      (PL:String-Rep (substr _str (+ (strlen _old) _pos 1)) _old _new))
;;; Рекурсивный вызов. В качестве первого аргумента выступает фрагмент
;;; исходной строки, начиная с конца первого вхождения искомой подстроки
;;; и до конца исходной строки.
    _str)
;;; Если искомая подстрока в исходной не обнаружена, то возвращается
;;; исходная строка без изменений.
)
;;; Пример использования:
;(PL:String-Rep "Тестовая строка" "ст" "*ЗАМЕНА*")
;;; "Те*ЗАМЕНА*овая *ЗАМЕНА*рока"
```

Применение рекурсивной функции с целью поиска и замены вхождений подстроки в строке удобно по той причине, что в данном случае каждый рекурсивный вызов производится с остатком исходной строки. Это дает возможность не создавать переменных для хранения промежуточных результатов. Кроме того, это не позволяет возникнуть коллизиям, когда замена искомой подстроки на новую подстроку порождает новое вхождение искомой подстроки. Поскольку обработка строк с очень большим количеством (несколько десятков тысяч) вхождений маловероятна, то даже некомпилированная функция не создает проблем с ограничением на количество рекурсивных вызовов.

10.1.2. Функция *PL:String->List*

Описание: разделяет строку на список строк по заданному разделителю.

Аргументы:

- <строка>, <разделитель> — строки.

Примечание

Передача аргументов другого типа приводит к ошибке.

Возврат: список строк. Текст функции см. в листинге 10.2.

Листинг 10.2. Функция `PL:String->List`

```
; Глава 10\Book01\PL_String-_List.lsp
(defun PL:String->List (_str _br / _pos)
  (if (setq _pos (vl-string-search _br _str))
;;; Если в строке обнаружено вхождение подстроки-разделителя,
;;; то запоминается его начальная позиция в переменной _pos.
    (cons (substr _str 1 _pos)
;;; Отрезается фрагмент исходной строки, от ее начала и до начала
;;; строки-разделителя. Функция cons ожидает возврата следующего
;;; рекурсивного вызова. По завершению этого вызова фрагмент исходной
;;; строки добавляется в начало возвращенного списка.
```

```
          (PL:String->List (substr _str (+ (strlen _br) _pos 1)) _br))
;;; Рекурсивный вызов. В качестве первого аргумента выступает фрагмент
;;; исходной строки, от конца первого вхождения строки-разделителя
;;; и до конца исходной строки.
    (list _str)
;;; Если строка-разделитель в исходной не обнаружена, то возвращается
;;; список, в который помещается исходная строка без изменений.
  ))
;;; Пример использования:
;(PL:String->List "Тест разделения  строки" " ")
;;; ("Тест" "разделения" "" "строки")
```

Данная функция очень похожа на предыдущую и приведена для демонстрации повторяемости рекурсивных решений. Основным достоинством этой функции является ее предельная простота и компактность.

10.1.3. Функция *PL:ReadFile*

Описание: считывает значимые строки из текстового файла.

Аргументы:

- <дескриптор файла>.

Примечание

Передача аргументов другого типа приводит к ошибке.

Возврат: список строк или `nil`. Текст функции показан в листинге 10.3.

Листинг 10.3. Функция `PL:ReadFile`

```
; Глава 10\Book01\PL_ReadFile.lsp
(defun PL:ReadFile (_file / _str)
  (if (setq _str (read-line _file))
;;; Если попытка считывания очередной строки из файла оказалась успешной,
;;; то переходим к дальнейшей обработке. В противном случае выходим
;;; из функции с возвратом nil.
    (if (= (setq _str (vl-string-trim " \t" _str)) "")
;;; Очистка строки от начальных и концевых пробелов и от табуляций.
      (PL:ReadFile _file)
;;; Если после очистки строка пустая, то переход к следующему вызову
;;; и по его завершению возврат результата без изменений.
      (cons _str (PL:ReadFile _file))
;;; Если строка не пустая, то ожидание возврата следующего вызова,
;;; по его завершению добавление строки в начало возвращенного списка
;;; и выход из функции с возвратом результата.
    )))
```

```
;;; Пример использования:
;(setq txtfile (open "D:\\Temp\\test.txt" "r"))
;;; #<file "D:\\Temp\\test.txt">
;(PL:ReadFile txtfile)
;;; ("while" "/C3\"math\"" "-1" ... "*" "+" "-" "// /" "// /=")
;(close txtfile)
```

Эта функция удобна компактностью, а также тем, что и считывание строк, и формирование результирующего списка строк происходят в естественном порядке.

10.1.4. Набор функций для работы с регулярными списками

Данный набор из десяти функций, текст которых приведен в листингах 10.4—10.13, предназначен для упрощения работы с регулярными списками, аналогичными матрицам. Могут обрабатываться списки следующего вида:

```
((1:1 1:2 ... 1:n)
 (2:1 2:2 ... 2:n)
        ...
 (m:1 m:2 ... m:n))
```

Для того чтобы не было путаницы с обычными списками, регулярные списки такого вида мы будем называть *матрицами*. Безусловно, такая аналогия не абсолютна, но для наших целей это не имеет решающего значения. Размер матриц ограничен возможностями AutoLISP по размеру списков и самовызову функций. Уровень вложенности матриц ограничен теми же факторами. Поскольку данный набор состоит из взаимосвязанных функций, то их назначение, аргументы и возвращаемые значения будут даны непосредственно в листингах.

Помимо иллюстрации рекурсии, этот набор можно рассматривать, как пример создания собственной библиотеки программиста для решения регулярно возникающих задач. Основной целью написания данного набора было упрощение обработки прямоугольных массивов точек. Тем не менее он может быть применим и в других случаях обработки регулярных списков.

Листинг 10.4. Функция `PL:MTX_Creat`

```
; Глава 10\Book01\PL_MTX_Creat.lsp
;;; Создает матрицу заданного формата и заполняет указанным значением.
;;; Связана с функцией PL:MList_Creat
;;; Формат вызова: (PL:MTX_Creat <ФОРМАТ> <ЗНАЧЕНИЕ>)
;;;        где: <ФОРМАТ>   - список вида (<INT> <INT> ...)
;;;             <ЗНАЧЕНИЕ> - любое допустимое в Автолиспе значение
```

```
;;; Пример: (PL:MTX_Creat '(4 3) nil)
;;; Возвращает: ((nil nil nil)
;;;              (nil nil nil)
;;;              (nil nil nil)
;;;              (nil nil nil))
(defun PL:MTX_Creat (_MTX _VAL)
  (PL:MList_Creat
;;; Вызов функции генерации списков.
    (car _MTX)
    (if (> (length _MTX) 1)
;;; Если не достигнут конец описания формата...
      (PL:MTX_Creat (cdr _MTX) _VAL)
;;; то следует рекурсивный вызов,
      _VAL
;;; в противном случае возвращается заданное значение.
    )))
```

Листинг 10.5. Функция `PL:MList_Creat`

```
; Глава 10\Book01\PL_MList_Creat.lsp
;;; Создает список указанного размера с заполнением заданным значением.
;;; Связана с функцией PL:MTX_Creat
;;; Формат вызова: (PL:MList_Creat <РАЗМЕР> <ЗНАЧЕНИЕ>)
;;;      где: <РАЗМЕР>   - любое число не меньше 1
;;;           <ЗНАЧЕНИЕ> - любое допустимое в Автолиспе значение
;;; Пример: (PL:MList_Creat 4 nil)
;;; Возвращает: (nil nil nil nil)
(defun PL:MList_Creat (_REP _VAL)
  (if (> _REP 1)
    (cons _VAL (PL:MList_Creat (1- _REP) _VAL))
    (list _VAL)))
```

Листинг 10.6. Функция `PL:MList_Set`

```
; Глава 10\Book01\PL_MList_Set.lsp
;;; Присваивает значение указанному элементу списка (список должен быть
;;; уже создан) и возвращает измененный список.
;;; Связана с функцией PL:MTX_Set
;;; Формат вызова: (PL:MList_Set <ЭЛЕМЕНТ> <СПИСОК> <ЗНАЧЕНИЕ>)
;;;      где: <ЭЛЕМЕНТ>  - номер заменяемого элемента
;;;           <СПИСОК>   - список
;;;           <ЗНАЧЕНИЕ> - любое допустимое в Автолиспе значение
;;; Пример: (PL:MList_Set 3 '(nil nil nil nil) T)
;;; Возвращает: (nil nil T nil)
```

```
(defun PL:MList_Set (_MLI _MTL _VAL)
  (if (> _MLI 1)
    (cons (car _MTL) (PL:MList_Set (1- _MLI) (cdr _MTL) _VAL))
    (cons _VAL (cdr _MTL))))
```

Листинг 10.7. Функция PL:MTX_Set

```
; Глава 10\Book01\PL_MTX_Set.lsp
;;; Присваивает значение указанному элементу матрицы (матрица должна быть
;;; уже создана) и возвращает измененную матрицу.
;;; Связана с функцией PL:MList_Set
;;; Формат вызова: (PL:MTX_Set <ЭЛЕМЕНТ> <МАТРИЦА> <ЗНАЧЕНИЕ>)
;;;      где: <ЭЛЕМЕНТ>  - список вида (<INT> <INT> ...), указывающий
;;;                        на заменяемый элемент
;;;           <МАТРИЦА>  - регулярный список
;;;           <ЗНАЧЕНИЕ> - любое допустимое в Автолиспе значение
;;; Пример: (PL:MTX_Set '(2 2) '((nil nil)
;;;                              (nil nil)
;;;                              (nil nil)
;;;                              (nil nil)) T)
;;; Возвращает: ((nil nil)
;;;              (nil T)
;;;              (nil nil)
;;;              (nil nil))
(defun PL:MTX_Set (_MTI _MTX _VAL)
  (PL:MList_Set (car _MTI) _MTX
;;; Вызов функции установки значения в списке.
    (if (> (length _MTI) 1)
;;; Если не достигнут конец списка, указывающего на заменяемый элемент...
      (PL:MTX_Set (cdr _MTI) (nth (1- (car _MTI)) _MTX) _VAL)
;;; то следует рекурсивный вызов,
      _VAL
;;; в противном случае возвращается заданное значение.
    )))
```

Листинг 10.8. Функция PL:MTX_Ev

```
; Глава 10\Book01\PL_MTX_Ev.lsp
;;; Возвращает значение указанного элемента матрицы.
;;; Простейшая рекурсивная функция.
;;; Формат вызова: (PL:MTX_Ev <ЭЛЕМЕНТ> <МАТРИЦА>)
;;;      где: <ЭЛЕМЕНТ> - список вида (<INT> <INT> ...), указывающий
;;;                       на искомый элемент
```

```
;;;            <МАТРИЦА> - регулярный список
;;; Пример: (PL:MTX_Ev '(2 2) '((11 12 13)
;;;                              (21 22 23)
;;;                              (31 32 33)
;;;                              (41 42 43)))
;;; Возвращает: 22
(defun PL:MTX_Ev (_MTI _MTX)
  (if (> (length _MTI) 1)
    (PL:MTX_Ev (cdr _MTI) (nth (1- (car _MTI)) _MTX))
    (nth (1- (car _MTI)) _MTX)))
```

Листинг 10.9. Функция `PL:MTX_Func`

```
; Глава 10\Book01\PL_MTX_Func.lsp
;;; Выполняет заданную функцию, используя в качестве аргумента элементы
;;; матрицы на уровне заданной глубины. Возвращает матрицу результатов.
;;; Формат вызова: (PL:MTX_Func <ФУНКЦИЯ> <МАТРИЦА> <УРОВЕНЬ>)
;;;      где: <ФУНКЦИЯ> - указатель функции или Лямбда-функция
;;;           <МАТРИЦА> - регулярный список
;;;           <УРОВЕНЬ> - уровень выполнения функции
;;; Пример: (PL:MTX_Func 'sin '((0.11 0.12)
;;;                              (0.21 0.22)
;;;                              (0.31 0.32)
;;;                              (0.41 0.42)) 2)
;;; Возвращает: ((0.109778 0.119712)
;;;              (0.208460 0.218230)
;;;              (0.305059 0.314567)
;;;              (0.398609 0.407760))
;;; Пример: (PL:MTX_Func 'length '((11 12 13)
;;;                                (21 22 23)
;;;                                (31 32 33)
;;;                                (41 42 43)) 1)
;;; Возвращает: (3 3 3 3)
(defun PL:MTX_Func (_FUNC _MTX _IN)
  (if (> _IN 1)
    (mapcar (function (lambda (_X) (PL:MTX_Func _FUNC _X (1- _IN))))
            _MTX
    )
    (mapcar _FUNC _MTX)))
```

Листинг 10.10. Функция `PL:MTX_Func2`

```
; Глава 10\Book01\PL_MTX_Func2.lsp
;;; Выполняет заданную функцию, используя в качестве аргументов элементы
;;; матриц на уровне заданной глубины. Возвращает матрицу результатов.
```

```
;;; Формат вызова: (PL:MTX_Func2 <ФУНКЦИЯ> <МАТРИЦА> <МАТРИЦА> <УРОВЕНЬ>)
;;;      где: <ФУНКЦИЯ> - указатель функции или Лямбда-функция
;;;           <МАТРИЦА> - регулярный список
;;;           <УРОВЕНЬ> - уровень выполнения функции
;;; Пример: (PL:MTX_Func2 '* '((0.11 0.12)
;;;                            (0.21 0.22)
;;;                            (0.31 0.32)
;;;                            (0.41 0.42))
;;;                         '((11 12)
;;;                            (21 22)
;;;                            (31 32)
;;;                            (41 42)) 2)
;;; Возвращает: ((1.21 1.44)
;;;              (4.41 4.84)
;;;              (9.61 10.24)
;;;              (16.81 17.64))
;;; Пример: (PL:MTX_Func2 'append '((0.11 0.12)
;;;                                 (0.21 0.22)
;;;                                 (0.31 0.32)
;;;                                 (0.41 0.42))
;;;                               '((11 12)
;;;                                 (21 22)
;;;                                 (31 32)
;;;                                 (41 42)) 1)
;;; Возвращает: ((0.11 0.12 11 12)
;;;              (0.21 0.22 21 22)
;;;              (0.31 0.32 31 32)
;;;              (0.41 0.42 41 42))
;;; Примечание: Функция легко может быть модернизирована для работы с
;;; любым количеством матриц. Единственным критерием является возможность
;;; приема аргументов Лямбда-функцией. При такой модернизации следует
;;; придерживаться формата именования функций: PL:MTX_Func#,
;;; где: # - количество матриц-аргументов, или давать таким функциям
;;; совершенно иные имена с целью недопущения конфликта имен.
(defun PL:MTX_Func2 (_FUNC _MTXA _MTXB _IN)
  (if (> _IN 1)
    (mapcar
      (function (lambda (_XA _XB) (PL:MTX_Func2 _FUNC _XA _XB (1- _IN))))
      _MTXA
      _MTXB
    )
    (mapcar _FUNC _MTXA _MTXB)))
```

Листинг 10.11. Функция PL:MTX_IDX

```
; Глава 10\Book01\PL_MTX_IDX.lsp
;;; Возвращает формат матрицы. Проверка формата проводится по первым
;;; элементам!
;;; Вызов: (PL:MTX_IDX <МАТРИЦА>)
;;;      где: <МАТРИЦА> - регулярный список
;;; Пример: (PL:MTX_IDX '((11 12 13)
;;;                       (21 22 23)
;;;                       (31 32 33)
;;;                       (41 42 43)))
;;; Возвращает: (4 3)
(defun PL:MTX_IDX (_MTX)
  (if (= (type (car _MTX)) 'list)
    (cons (length _MTX) (PL:MTX_IDX (car _MTX)))
    (list (length _MTX))))
```

Листинг 10.12. Функция PL:MList_Func

```
; Глава 10\Book01\PL_MList_Func.lsp
;;; Выполняет заданную функцию, используя в качестве аргумента заданный
;;; элемент списка. Возвращает измененный список.
;;; Формат вызова: (PL:MList_Func <ФУНКЦИЯ> <ЭЛЕМЕНТ> <СПИСОК>)
;;;      где: <ФУНКЦИЯ> - указатель функции или Лямбда-функция
;;;           <ЭЛЕМЕНТ> - номер заменяемого элемента
;;;           <СПИСОК>  - список
;;; Пример: (PL:MList_Func 'sin 3 '(0.1 0.2 0.3 0.4 0.5))
;;; Возвращает: (0.1 0.2 0.29552 0.4 0.5)
(defun PL:MList_Func (_FUNC _MLI _MTL)
  (if (> _MLI 1)
    (cons (car _MTL) (PL:MList_Func _FUNC (1- _MLI) (cdr _MTL)))
    (cons (apply _FUNC (list (car _MTL))) (cdr _MTL))))
```

Листинг 10.13. Функция PL:MTI_Func

```
; Глава 10\Book01\PL_MTI_Func.lsp
;;; Выполняет заданную функцию, используя в качестве аргумента заданный
;;; элемент матрицы. Возвращает измененную матрицу.
;;; Формат вызова: (PL:MTI_Func <ФУНКЦИЯ> <ЭЛЕМЕНТ> <МАТРИЦА>)
;;;      где: <ФУНКЦИЯ> - указатель функции или Лямбда-функция
;;;           <ЭЛЕМЕНТ> - список вида (<INT> <INT> ...), указывающий
;;;                       на искомый элемент
```

```
;;;            <МАТРИЦА> - регулярный список
;;; Пример: (PL:MTI_Func 'sin '(3 2) '((0.11 0.12)
;;;                                    (0.21 0.22)
;;;                                    (0.31 0.32)
;;;                                    (0.41 0.42)))
;;; Возвращает: ((0.11 0.12)
;;;              (0.21 0.22)
;;;              (0.31 0.305059)
;;;              (0.41 0.42))
(defun PL:MTI_Func (_FUNC _MTI _MTX)
  (PL:MList_Set
    (car _MTI)
    _MTX
    (if (> (length _MTI) 1)
      (PL:MTI_Func _FUNC (cdr _MTI) (nth (1- (car _MTI)) _MTX))
      (apply _FUNC (list (car _MTX))))))
```

10.2. Выбор примитивов методами ActiveX

Одним из распространенных действий при программировании в среде AutoCAD является выбор примитивов как с участием пользователя, так и полностью программным путем. Если с выбором, использующим функции `ssget`, `entsel` и `nentsel`, большинство LISP-программистов, как правило, знакомы, то выбор с помощью средств ActiveX нередко остается вне поля зрения программистов.

Для LISP-программирования наиболее типичными являются два случая, когда использование ActiveX может быть целесообразно. Это программный выбор примитивов по условию (фильтру) и организация пользовательского выбора, в том числе с наложением условий выбора. Для этой цели используются методы `Select` и `SelectOnScreen`. Их назначение и аргументы описаны в *главе 7*, а здесь мы рассмотрим примеры практического использования.

В отличие от традиционного для AutoLISP механизма создания наборов примитивов, при использовании ActiveX существует возможность создавать и манипулировать несколькими наборами одновременно. Связано это с тем, что имеется активный безымянный набор, доступ к которому можно получить так:

```
(setq sel (vla-get-ActiveSelectionSet
  (vla-get-ActiveDocument (vlax-get-acad-object))))
;;; #<VLA-OBJECT IAcadSelectionSet 06174f54>
```

Кроме того, есть возможность создания именованных наборов, каждый из которых может содержать любые примитивы, в том числе и входящие в другие наборы. Создаются именованные наборы следующим образом (типичный пример):

```
(setq doc  (vla-get-ActiveDocument (vlax-get-acad-object))
;;; получили активный документ
```

```
      snam "pl-temp-selection"
;;; сохранили в переменной имя будущего набора
      sels (vla-get-SelectionSets doc))
;;; получили семейство наборов
(if (vl-catch-all-error-p
      (setq sel (vl-catch-all-apply (function vla-item)
        (list sels snam))))
;;; попытались из семейства наборов получить набор с ранее заданным
;;; именем, с перехватом ошибки, возникающей при отсутствии набора
   (setq sel (vla-Add sels snam)))
;;; при возникновении ошибки получения набора, создали его
(vla-Clear sel)
;;; очистили полученный набор (если нужно)
```

Когда набор получен, в него можно добавлять графические объекты. Механизм добавления примитивов одинаков для всех видов наборов, поэтому примеры будут даны с использованием безымянного набора. Тем не менее разница между именованными наборами и безымянным существует. Заключается она в несколько большей, чем у именованных наборов, склонности безымянного набора к возникновению ошибок. Эти особенности, по-видимому, вызваны какими-то нюансами реализации ActiveX и плохо поддаются прогнозированию. В общем случае можно рекомендовать использовать безымянный набор для оперативной манипуляции объектами, без их удаления и с обязательной очисткой набора по окончании манипуляций. Если предполагается сложная обработка, то лучше воспользоваться именованными наборами. Для обоих видов наборов нежелательны такие манипуляции с входящими в них примитивами, которые могут приводить к их (примитивов) исчезновению. Если алгоритм программы предполагает возможность удаления объектов, входящих в набор, то предпочтительнее такое решение, когда сначала создается список из объектов, затем набор очищается и только после этого производятся манипуляции с примитивами.

Выглядит это примерно так:

```
(setq sel (vla-get-ActiveSelectionSet
            (vla-get-ActiveDocument (vlax-get-acad-object))))
;;; Получили безымянный набор.
(vla-Clear sel) ;_ Очистили полученный набор.
(pl:obj-filter-select-all sel '((8 . "0"))) ;_ Сделали выбор.
;;; Функция pl:obj-filter-select-all описана ниже.
(vlax-for i sel (setq slist (cons i slist))) ;_ Создали список.
(vla-Clear sel) ;_ Очистили набор.
(foreach i slist (...)) ;_Обработка примитивов.
```

На примерах мы разберем два наиболее распространенных варианта выбора: интерактивный и программный по всему документу, оба с возможностью установки ограничительных фильтров.

10.2.1. Интерактивный выбор

Такой выбор применяется тогда, когда необходимо, чтобы пользователь сам указал примитивы, которые должны попасть в набор. Соответствующий текст приведен в листинге 10.14.

Листинг 10.14. Функция `pl:obj-filter-select-manual`

```
; Глава 10\Book02\pl_obj-filter-select-manual.lsp
;;; Помещает выбранные пользователем примитивы в набор, переданный
;;; в качестве первого аргумента. Возвращает nil.
;;; Формат вызова: (pl:obj-filter-select-manual <НАБОР> <ФИЛЬТР>)
;;;      где: <НАБОР>  - VLA-объект
;;;           <ФИЛЬТР> - список фильтров с синтаксисом функции ssget
;;; Пример: (setq sel (vla-get-ActiveSelectionSet
;;;          (vla-get-ActiveDocument (vlax-get-acad-object))))
;;;         (pl:obj-filter-select-manual sel '((8 . "*")))
(defun pl:obj-filter-select-manual (sel filter / i)
  (vla-SelectOnScreen sel (vlax-safearray-fill (vlax-make-safearray
      vlax-vbInteger (setq i (cons 1 (length filter))))
      (mapcar (function car) filter))
    (vlax-safearray-fill (vlax-make-safearray vlax-vbVariant i)
      (mapcar (function cdr) filter))))
```

Функция `pl:obj-filter-select-manual` требует обязательной передачи фильтра, в качестве второго аргумента. В том случае, когда ограничений выбора пользователя быть не должно, можно применить такой фильтр, соответствующими которому оказываются все примитивы, например: `'((8 . "*"))`.

10.2.2. Программный выбор

В отличие от интерактивного, программный выбор не требует взаимодействия с пользователем. Выбор происходит исключительно на основе критериев, заданных фильтрами. Текст функции приведен в листинге 10.15.

Листинг 10.15. Функция `pl:obj-filter-select-all`

```
; Глава 10\Book02\pl_obj-filter-select-all.lsp
;;; Помещает выбранные пользователем примитивы в набор, переданный
;;; в качестве первого аргумента. Возвращает nil.
;;; Формат вызова: (pl:obj-filter-select-manual <НАБОР> <ФИЛЬТР>)
;;;      где: <НАБОР>  - VLA-объект
;;;           <ФИЛЬТР> - список фильтров с синтаксисом функции ssget
;;; Пример: (setq sel (vla-get-ActiveSelectionSet
;;;          (vla-get-ActiveDocument (vlax-get-acad-object))))
;;;         (pl:obj-filter-select-all sel '((8 . "0")))
```

```
(defun pl:obj-filter-select-all (sel filter / i)
  (vla-Select sel acSelectionSetAll nil nil (vlax-safearray-fill
   (vlax-make-safearray vlax-vbInteger (setq i (cons 1 (length filter))))
      (mapcar (function car) filter))
   (vlax-safearray-fill (vlax-make-safearray vlax-vbVariant i)
      (mapcar (function cdr) filter))))
```

Функция `pl:obj-filter-select-manual`, как и предыдущая, требует обязательной передачи фильтра в качестве второго аргумента. Для программного выбора использование фильтров является типичным, тем не менее в случае, когда необходимо выбрать все содержимое документа, можно применить фильтр, при котором оказываются выбранными все примитивы, например: `'((8 . "*"))`.

10.3. Особенности обработки семейств объектов

При обработке семейств объектов, в том числе наборов, могут возникать некоторые моменты, неочевидные на первый взгляд. Практически всегда это бывает связано с самим доступом к объектам, членам семейства. Существует несколько способов такого доступа.

Первый из способов — это доступ по индексу при помощи метода `Item`. В этом случае в качестве аргумента передается индекс (номер) искомого члена коллекции, считая с нуля. Синтаксис функции, реализующей данный метод, выглядит так:

```
(vla-Item <семейство> <индекс>)
```

При таком доступе к членам семейства нужно помнить о том, что каждый вызов метода `Item` автономен и индексы могут измениться за время, прошедшее с момента предыдущего обращения. Это может произойти, например, в результате удаления объектов. Таким образом, осуществлять проход по семейству данным способом можно только в том случае, когда есть полная уверенность, что с момента предыдущего обращения не производились операции удаления и добавления объектов обрабатываемого семейства.

Второй способ основан на том, что некоторые объекты имеют свойство `Name`, а метод `Item` может использовать это свойство для доступа к объекту. Синтаксис функции не меняется:

```
(vla-Item <семейство> <имя>)
```

Только вместо индекса фигурирует имя объекта.

В этом случае для доступа к объекту надо знать его имя. Поскольку имена объектов самопроизвольно не изменяются, то и проблемы с получением конкретного объекта не возникает. Единственное обстоятельство, о котором надо знать при

использовании именного доступа, — это то, что некоторые семейства могут содержать одновременно объекты, как имеющие имя, так и не имеющие такого свойства.

Третий способ обращения к объектам семейства использует индексный доступ, но не к одному объекту, а ко всем членам семейства. Для этого используется циклическая функция `vlax-for`. Функция обладает несложным синтаксисом:

```
(vlax-for <переменная> <семейство> (...))
```

Несмотря на то, что обращение фактически происходит последовательно, с точки зрения программиста, можно считать его одномоментным, поскольку индексы объектов выбираются при обращении функции к семейству один раз, а собственно доступ происходит позже. Такой алгоритм обращения вносит некоторые особенности в доступ. Очень нежелательно удаление объектов обрабатываемого семейства, иных, чем обрабатываемый объект на каждом шаге цикла. Причина в том, что удаление объекта, обращения к которому еще не было, приводит к ошибке в момент, когда обращение происходит.

Кроме нежелательности удаления членов семейства, есть закономерная особенность и в реакции на появление новых объектов семейства внутри `vlax-for`. Собственно, вся реакция сводится к ее отсутствию. Объекты, добавленные в семейство, внутри цикла не порождают новых циклов обработки. Таким образом, нельзя вызвать эту функцию в бесконечном цикле. Только для примера в листинге 10.16 приведем демонстрационную функцию, которая обрабатывает отрезки (предварительно удалив примитивы других типов) и добавляет новые отрезки в пространство модели.

Листинг 10.16. Функция `pl_demo`

```
; Глава 10\Book03\pl_demo.lsp
(defun pl_demo (/ model count startp)
  (setq model (vla-get-ModelSpace
                (vla-get-ActiveDocument (vlax-get-acad-object))))
;;; Получили доступ к пространству модели.
  (if (zerop (vla-get-Count model)) ;_ если нет примитивов...
    (princ "/nNo objects in Model Space!") ;_ сообщаем и выходим
    (progn (vlax-for i model
;;; в противном случае удаляем все примитивы, не являющиеся
;;; отрезками (для простоты демонстрации)
        (if (/= "AcDbLine" (vla-get-ObjectName i))
          (vla-Delete i)))
      (if (zerop (setq count (vla-get-Count model)))
;;; если после удаления неотрезков модель не содержит примитивов...
        (princ "/nNo lines in Model Space!") ;_ сообщаем и выходим
        (progn (setq startp (vla-get-EndPoint (vla-Item model
               (1- count))))
```

```
;;; иначе выбираем точку из координат последнего отрезка в наборе.
            (vlax-for line model ;_ проверяемый вызов vlax-for
            (vla-AddLine model startp (vla-get-StartPoint line))
;;; добавляем новый отрезок в пространство модели, используя
;;; координаты уже имеющихся отрезков
            (setq startp (vla-get-EndPoint line)))))))
;;; получаем новую точку для начала следующего отрезка
  (princ))
```

Эта функция не производит какой-то осмысленной работы и предназначена исключительно для демонстрации алгоритма функции `vlax-for`. В результате ее деятельности окажутся стертыми все примитивы пространства модели, не являющиеся отрезками, а все отрезки окажутся соединенными между собой новыми отрезками. Если бы происходило неоднократное обращение к индексу семейства, то эта функция неизбежно попала бы в бесконечный цикл.

10.4. Пример работы с внешним приложением (чтение файла Excel 2003)

В этом разделе разберем известный пример, который показывает, как работать с другими приложениями, имеющими свою объектную модель. С помощью COM-технологии можно устанавливать связь с нужными объектами других приложений и выполнять необходимые действия с использованием "чужих" методов и свойств.

Довольно часто приходится в рисунке AutoCAD создавать таблицу (например, спецификации), данные для которой находятся в файле Excel. В книге "AutoCAD 2002" серии "В подлиннике" был приведен пример LISP-программы переноса данных из Excel 2000 в таблицу, создаваемую отрезками в рисунке AutoCAD 2002. Результаты обсуждения этого примера в Интернете были очень полезны. В частности, стала понятной проблема корректной выгрузки табличного процессора Excel из памяти (возможны случаи, когда программы, используемые в качестве COM-серверов, множатся в оперативной памяти).

В другой книге серии "В подлиннике" ("AutoCAD 2004: разработка приложений и адаптация") показаны примеры импорта таблицы из Excel 2002 и экспорта данных из рисунка AutoCAD в таблицу Excel 2002.

Сейчас разберем пример решения такой задачи для Excel 2003 с построением таблицы в виде примитива ACAD_TABLE. Напомним, что корневым элементом внешнего приложения является объект `Application`, создать который в памяти можно с помощью функций `vlax-create-object`, `vlax-get-object` или `vlax-get-or-create-object`.

10.4.1. Анализ методов и свойств приложения Excel

Определим с помощью реестра, под каким именем следует в COM-технологии обращаться к приложению Microsoft Excel 2003. В результате получим строку "Excel.Application.11" (можно пользоваться и укороченной строкой "Excel.Application" — в этом случае операционная система сама определяет, какая версия Excel является последней на данном компьютере, а программист должен учитывать возможность вызова не той версии, на которую он рассчитывал).

С помощью функции `vlax-dump-object` прочитаем список методов, которые можно применять к объекту приложения Excel, и максимально допустимое количество аргументов этих методов. Делается это с помощью следующего выражения:

```
(vlax-dump-object (vlax-get-or-create-object "Excel.Application.11") T)
```

Получим список свойств и методов изучаемого объекта. Перечислим сначала некоторые методы (на самом деле они располагаются в конце листинга):

`ActivateMicrosoftApp` (1), `AddChartAutoFormat` (3), `AddCustomList` (2), `Calculate` (), `CalculateFull` (), `CalculateFullRebuild` () и т. д.

Одновременно получим и названия свойств (в этом списке фигурируют и дочерние семейства):

`ActiveCell`, `ActiveChart`, `ActivePrinter`, `ActiveSheet`, `ActiveWindow`, `ActiveWorkbook`, `AddIns`, `AlertBeforeOverwriting`, `AltStartupPath`, `AnswerWizard`, `Application`, `ArbitraryXMLSupportAvailable`, `AskToUpdateLinks`, `Assistant`, `AutoCorrect`, `AutoFormatAsYouTypeReplaceHyperlinks`, `AutomationSecurity`, `AutoRecover`, `Build`, `CalculateBeforeSave`, `Calculation`, `CalculationInterruptKey`, `CalculationState`, `CalculationVersion` и т. д.

10.4.2. Тестовый файл книги Excel

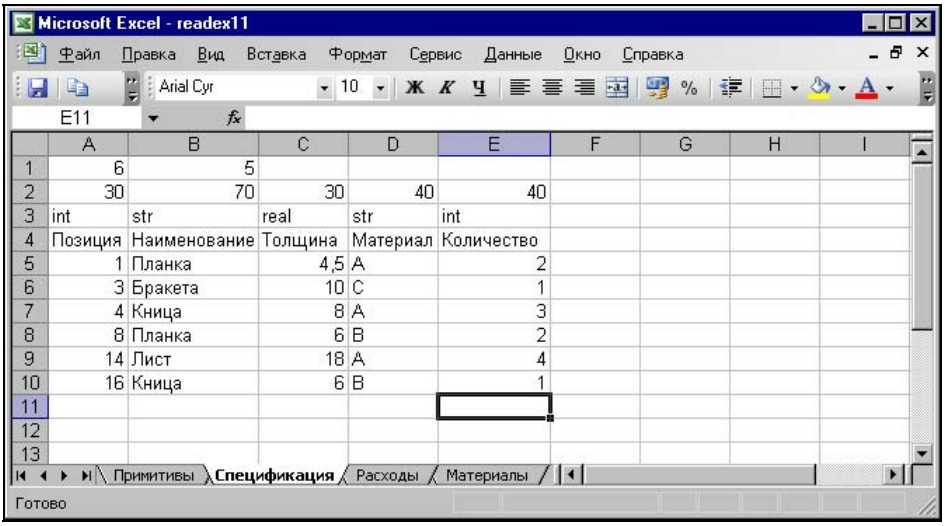
Предположим, что данные для таблицы спецификации, которую нам нужно начертить в рисунке AutoCAD, располагаются в листе **Спецификация** книги (файла) d:\r 16\readex11.xls (рис. 10.1).

На рисунке видно, что в нашем файле четыре листа (помимо листа **Спецификация**, есть еще листы **Примитивы**, **Расходы**, **Материалы**), поэтому программа должна не только открыть книгу Excel, но еще и обратиться к нужному листу.

В листе **Спецификация** для облегчения программирования введены дополнительные данные, которые уточняют характеристики будущей таблицы деталей, создаваемой в рисунке AutoCAD.

В первой строке в ячейке A1 дано количество деталей (т. е. строк таблицы без учета заголовка — `6`), а в ячейке B1 — количество столбцов таблицы (`5`). Вторая строка листа **Спецификация** — размеры столбцов по ширине в мм (`30`, `70`, `30`, `40` и `40`). Третья строка уточняет тип данных, которые располагаются в соответст-

вующем столбце (`int` — целое, `real` — вещественное, `str` — текст). При выводе чисел будем давать по одной цифре после десятичной точки и прижимать их в ячейках вправо.



	A	B	C	D	E
1	6	5			
2	30	70	30	40	40
3	int	str	real	str	int
4	Позиция	Наименование	Толщина	Материал	Количество
5	1	Планка	4,5	A	2
6	3	Бракета	10	C	1
7	4	Кница	8	A	3
8	8	Планка	6	B	2
9	14	Лист	18	A	4
10	16	Кница	6	B	1
11					
12					
13					

Рис. 10.1. Окно просмотра листа книги readex11.xls

Четвертая строка — названия колонок таблицы (`Позиция`, `Наименование`, `Толщина`, `Материал`, `Количество`). Все текстовые данные при выводе в ячейки будем прижимать влево.

Сделаем аргументами нашей программы следующие:

- имя XLS-файла (например, d:\r 16\readex11.xls);
- название листа в файле (например, **Спецификация**);
- высоту строк таблицы (например, 5 мм);
- высоту букв в таблице (например, 3 мм).

10.4.3. Тексты программ

В листингах 10.17—10.19 приведен текст основной функции `readex11_com` и тексты вспомогательных функций.

Листинг 10.17. Функция `readex11_com`

```
; Глава 10\Book04\readex11_com.lsp
;=================================================
; Программа COM-связи AutoCAD - Microsoft Excel 2003
;=================================================
```

```
; Вычерчивание таблицы по данным из книги Excel
; Используются текущий слой, текущий цвет и создаваемый стиль таблиц
;
; Аргументы:
;   tb_xls - имя файла книги Excel;
;   sheetname - имя листа;
;   hcell - высота ячейки таблицы;
;   hlet - высота букв текста таблицы;
;   show - признак показа Excel во время работы (T или nil).
;
; Глобальные переменные:
;   g_oex - VLA-объект приложения Excel 2003;
;   g_wkbs - VLA-объект семейства Workbooks;
;   g_awb - VLA-объект активной книги;
;   g_shs - VLA-объект семейства Sheets;
;   g_mainsh - VLA-объект нужного листа книги;
;   g_cell - VLA-объект ячейки.
;
; Примеры обращения к программе:
; (readex11_com "d:\\r 16\\readex11.xls" "Спецификация" 8.0 5.0 T)
; (readex11_com "d:\\r 16\\readex11.xls" "Спецификация" 5.0 3.0 nil)
;
(defun readex11_com (tb_xls sheetname hcell hlet show / nlines ncolumns
  table_items i j row pt0 widths types headers val msp tbl old_echo)
; Подгрузка библиотек, обеспечивающих работу с ActiveX
(vl-load-com)
; Отключение эхо-вывода
(setq old_echo (getvar "CMDECHO"))
(setvar "CMDECHO" 0)
;
; Установление связи с Excel
(ex11_set_connect show)
; Указатель семейства Workbooks
(setq g_wkbs (vlax-get-property g_oex "Workbooks"))
; Открытие файла (книги) и получение указателя книги
(setq g_awb (vlax-invoke-method g_wkbs "Open" tb_xls))
; Проверка открытия файла
(if (not g_awb)
  (progn
    (alert (strcat "Не обнаружен файл " tb_xls))
    (ex_break_connect)
    (exit)
  )
)
```

```
; Чтение списка листов; не обязательно активизировать нужный
(setq g_shs (vlax-get-property g_awb "Worksheets"))
; Указатель на лист с нужным именем
(vlax-for s g_shs (if (= sheetname (vlax-get-property s "Name"))(setq g_mainsh
s)))
(if (not g_mainsh)
  (progn
    (alert (strcat "Не обнаружен лист " sheetname))
    (ex_break_connect)
    (exit)
  )
)
;
; Чтение количества строк в будущей таблице спецификации
;  (из ячейки A1 листа Excel), без учета строки заголовков
(setq g_cell
  (vlax-variant-value (vlax-invoke-method g_mainsh "Evaluate" "A1")))
(setq nlines
  (fix (vlax-variant-value (vlax-get-property g_cell "Value"))))
(vlax-release-object g_cell)
; Чтение количества столбцов в будущей таблице спецификации
;  (из ячейки B1 листа Excel)
(setq g_cell
  (vlax-variant-value (vlax-invoke-method g_mainsh "Evaluate" "B1")))
(setq ncolumns
  (fix (vlax-variant-value (vlax-get-property g_cell "Value"))))
(vlax-release-object g_cell)
;
; Получение списка элементов таблицы (ряды 2,3,4, ... Excel)
; Ряд 2 - ширина столбцов в мм
; Ряд 3 - заголовки столбцов
(setq table_items nil j 1)
(repeat (+ nlines 3)
  (setq row nil j (1+ j) i -1)
  (repeat ncolumns
    (setq i (1+ i))
    (setq g_cell
      (vlax-variant-value
        (vlax-invoke-method
          g_mainsh
          "Evaluate"
          (strcat (chr (+ i (ascii "A"))) (itoa j))
        )
      )
    )
```

```
    (setq row
      (append
        row
        (list
          (vlax-variant-value
            (vlax-get-property g_cell "Value")
          )
        )
      )
    )
    (vlax-release-object g_cell)
  );repeat ncolumns
  (setq table_items (append table_items (list row)))
);repeat nlines
;
; Закрытие и выгрузка Excel
(ex_break_connect)
;
; Запрос точки левого верхнего угла таблицы
(setq pt0 nil)
(while (null pt0)
  (setq pt0 (getpoint "\nТочка левого верхнего угла таблицы: "))
);while
;
(setq widths (nth 0 table_items) types (nth 1 table_items))
;
(setq msp (vla-get-ModelSpace (vla-get-activeDocument
  (vlax-get-acad-object))))
; Создание стандартной таблицы с колонками равной ширины
(setq tbl (vla-AddTable msp (vlax-3d-point pt0) (+ 2 nlines)
  ncolumns hcell (car widths)))
; Корректировка ширины столбцов
(setq i -1)
(repeat ncolumns
  (setq i (1+ i))
  (vla-SetColumnWidth tbl i (nth i widths))
)

; Заголовок таблицы
; Высота букв
(vla-SetTextHeight tbl acTitleRow hlet)
; Текст
(vla-SetText tbl 0 0 "Спецификация")
;
; Заголовки колонок
; Высота букв
(vla-SetTextHeight tbl acHeaderRow hlet)
```

```
(setq headers (nth 2 table_items) i -1)
(repeat ncolumns
  (setq i (1+ i))
  (vla-SetText tbl 1 i (nth i headers))
);
;
; Вписывание элементов таблицы
; Высота букв
(vla-SetTextHeight tbl acDataRow hlet)
(setq j 2)
(repeat nlines
  (setq j (1+ j) i -1)
  (repeat ncolumns
    (setq i (1+ i) val (nth i (nth j table_items)))
    (if (= "str" (nth i types))
      (progn
        (vla-SetText tbl (1- j) i val)
        ; Выравнивание в ячейке
        (vla-SetCellAlignment tbl (1- j) i acBottomLeft)
      )
      (progn
        (vla-SetText tbl (1- j) i (rtos val 2 1))
        ; Выравнивание в ячейке
        (vla-SetCellAlignment tbl (1- j) i acBottomRight)
      )
    )
  );repeat ncolumns
);repeat nlines
;
; Восстановление режима эхо-вывода
(setvar "CMDECHO" old_echo)
;
(redraw)
(princ)
);defun readex11_com
```

Листинг 10.18. Функция ex11_set_connect

```
; Глава 10\Book04\ex11_set_connect.lsp
; Установка связи с Excel 2003
; Аргумент:
;   vis = T - сделать Excel видимым;
;   vis = nil - сделать Excel невидимым.
;
```

```
(defun ex11_set_connect (vis / )
  (setq g_oex (vlax-get-or-create-object "Excel.Application.11"))
; Если связь не установлена, то аварийно завершить работу
  (if (null g_oex)
    (progn
      (alert "Невозможно запустить Microsoft Excel 2003")
      (exit)
    )
  );if
; Сделать Excel видимым в зависимости от параметра vis
 (if vis (vlax-put-property g_oex "Visible" :vlax-true))
);defun ex11_set_connect
```

Листинг 10.19. Функция ex_break_connect

```
; Глава 10\Book04\ex11_break_connect.lsp
; Разрыв связи с Excel и выгрузка из памяти
(defun ex_break_connect ( / )
(vlax-invoke-method g_oex "Quit")
; Освобождаем объекты, связанные с Excel,
; для корректной выгрузки Excel из памяти
(mapcar
  (function (lambda (x)
    (if
      (and x (not (vlax-object-released-p x)))
      (vlax-release-object x)
   )
 ))
  (list g_cell g_mainsh g_shs g_awb g_wkbs g_oex)
)
; Сборка мусора
(setq g_cell nil g_mainsh nil g_shs nil g_awb nil g_wkbs nil g_oex nil)
(gc)
);defun ex11_break_connect
```

Перед выполнением программ, приведенных в листингах 10.17—10.19, необходимо их загрузить, а затем обратиться к головной функции `readex11_com`, например:

```
(readex11_com "d:\\r 16\\readex11.xls" "Спецификация" 8.0 5.0 T)
(readex11_com "d:\\r 16\\readex11.xls" "Спецификация" 5.0 3.0 nil)
```

Результат построения таблицы в первом случае приведен на рис. 10.2, во втором — на рис. 10.3.

Спецификация				
Позиция	Наименование	Толщина	Материал	Количество
1	Планка	4,5	A	2
3	Бракета	10	C	1
4	Кница	8	A	3
8	Планка	6	B	2
14	Лист	18	A	4
16	Кница	6	B	1

Рис. 10.2. Построенная таблица (`hcell` = 8.0, `hlet` = 5.0)

Спецификация				
Позиция	Наименование	Толщина	Материал	Количество
1	Планка	4,5	A	2
3	Бракета	10	C	1
4	Кница	8	A	3
8	Планка	6	B	2
14	Лист	18	A	4
16	Кница	6	B	1

Рис. 10.3. Построенная таблица (`hcell` = 5.0, `hlet` = 3.0)

10.4.4. Анализ программ

Дадим некоторые пояснения к работе комплекса программ.

После установки связи с Microsoft Excel 2003 головная программа вычисляет последовательно указатели: `g_oex` (указатель объекта самого приложения Excel), `g_wkbs` (указатель семейства `Workbooks`), `g_awb` (указатель активной книги после открытия в Excel файла readex11.xls), `g_shs` (указатель семейства листов), `g_mainsh` (указатель нужного листа, в нашем примере — **Спецификация**).

Затем с помощью метода `Evaluate` читаются значения, записанные в листе **Спецификация** в ячейках A1 (`nlines` — количество строк или деталей) и B1 (`ncolumns` — количество столбцов в будущей таблице). Поскольку метод `Evaluate` возвращает значение типа `VARIANT`, для его сохранения в виде обычного целого числа применяется преобразование типа с помощью функции `vlax-variant-value`.

Далее, применяя тот же метод `Evaluate`, сканируем строки 2—10 листа **Спецификация** и получаем список `table_items` из девяти элементов, каждый из которых является списком, состоящим из пяти значений:

```
((30.0 70.0 30.0 40.0 40.0) ("int" "str" "real" "str" "int")
("Позиция" "Наименование" "Толщина" "Материал" "Количество")
(1.0 "Планка" 4.5 "A" 2.0) (3.0 "Бракета" 10.0 "C" 1.0)
(4.0 "Кница" 8.0 "A" 3.0) (8.0 "Планка" 6.0 "B" 2.0)
(14.0 "Лист" 18.0 "A" 4.0) (16.0 "Кница" 6.0 "B" 1.0)).
```

Первый элемент списка table_items (ширина столбцов) сохраняется для удобства в переменной widths, второй (типы значений по столбцам) — в переменной types, а третий (заголовки столбцов) — в переменной headers.

Из списка видно, что все значения, являющиеся как вещественными, так и целыми числами, представлены в вещественном виде. Поэтому при вписывании их в таблицу нужно будет пользоваться типами "int" или "real".

После формирования table_items программа ex_break_connect закрывает соединение с табличным процессором Excel и выгружает его из памяти. При этом проверяется, какие глобальные переменные, содержащие указатели объектов Excel, сформированы, и их объекты аннулируются с помощью функции vlax-release-object. Это позволяет избежать неприятного эффекта, когда приложение Excel не удаляется из памяти. При повторениях процесса количество неудаленных приложений растет и может даже исчерпать всю оперативную память на компьютере.

Далее интерактивно запрашивается точка p0, которая будет точкой верхнего левого угла таблицы спецификации деталей. Цикл, организованный с помощью функции while, исключает возможность пустого ввода нажатием на клавишу <Enter> (это можно было сделать и путем применения функции initget).

Сначала с помощью метода AddTable строится пустая таблица с колонками одинаковой ширины. Затем с помощью метода SetColumnWidth задаются нужные значения ширины.

Заполнение ячеек выполняется в три шага:

- С помощью метода SetTextHeight задается высота букв текста.
- С помощью метода SetText задается сам текст.
- С помощью метода SetCellAlignment задается выравнивание в ячейке.

Для заголовков таблицы и колонок дополнительное выравнивание не выполняется, поскольку используемое по умолчанию выравнивание acMiddleCenter нас вполне устраивает. В методе SetTextHeight в качестве аргумента должен задаваться тип ячеек (он принимает значения: acTitleRow для заголовка таблицы, acHeaderRow для заголовков колонок и acDataRow — для ячеек с данными).

Приведенные в листингах 10.17—10.19 программы удобны тем, что в самом файле Excel можно задать такие настройки таблицы, как количество и тип элементов, ширина столбцов. Читатель может на основе этого текста создать для себя наиболее удобный вариант. Опытный программист может добавить к программе более тщательный разбор значений, занесенных предварительно в книгу Excel, поскольку при больших объемах данных обязательно возникают ошибки, которые надо обрабатывать.

Послесловие

Уважаемый читатель!

В данной книге мы постарались максимально полно осветить вопросы и проблемы программирования в среде AutoCAD с применением языков AutoLISP и Visual LISP, а также инструментов взаимодействия с DCL-окнами, меню и справочными системами. Как вы, наверное, заметили, в книге имеются материалы, которые ранее в таком объеме никогда не излагались в российской и иностранной печати. Авторы далеки от мысли, что весь приведенный исходный код свободен от недостатков, однако мы приложили максимум усилий для того, чтобы избежать ошибок. Поэтому надеемся, что настоящая книга будет вам полезна.

За информацией о дальнейших планах мы приглашаем вас на сайты **www.private.peterlink.ru/poleshchuk/cad**, **www.cad.dp.ua/poleschuk.html**, **www.uniip.ru**, **poleshchuk.spb.ru**.

Ваши замечания, пожелания, обнаруженные ошибки и опечатки просим направлять по электронной почте на адреса авторов: **poleshchuk@peterlink.ru** и **alaspher@gmail.com**.

ПРИЛОЖЕНИЕ 1

Алфавитный перечень функций

В табл. П1.1 приведен перечень всех функций языка LISP, их назначение и номера разделов, в которых эти функции описаны. В таблице не присутствуют функции методов и свойств, имеющие префиксы `vla-`, `vla-get-` и `vla-put-` и рассмотренные в *главах 7* и *8* (включены только три часто используемые функции с префиксом `vla-get-`).

Помимо упомянутых функций, в каждой конкретной версии системы AutoCAD могут подгружаться и другие, расположенные во вспомогательных файлах. Для получения сведений о всех загруженных в данный момент функциях можно воспользоваться специальной программой, расположенной в файле acadinfo.lsp папки Support системы AutoCAD. Необходимо загрузить файл (например, с помощью выражения `(load "acadinfo")`) и выполнить команду ACADINFO. Эта команда попросит указать имя TXT-файла, в который следует записать собранную системную информацию.

Таблица П1.1. *Список функций и соответствующих разделов с описанием*

Функция	**Назначение**	**Номер раздела**
+	Сложение	1.4.1
–	Вычитание	1.4.2
*	Умножение	1.4.3
/	Деление	1.4.4
=	Равенство	1.5.1
/=	Неравенство	1.5.2
<	Меньше	1.5.3
<=	Меньше или равно	1.5.4

Таблица П1.1 *(продолжение)*

Функция	**Назначение**	**Номер раздела**
>	Больше	1.5.5
>=	Больше или равно	1.5.6
~	Дополнение до –1	1.4.7
1+	Увеличение на 1	1.4.5
1-	Уменьшение на 1	1.4.6
abs	Абсолютная величина	1.4.8
acad-pop-dbmod	Восстанавливает значение системной переменной DBMOD	1.18.14
acad-push-dbmod	Запоминает значение системной переменной DBMOD	1.18.13
acad_colordlg	Вызов диалогового окна с палитрой цветов	1.10.19
acad_helpdlg	Вызов справки	5.3.1
acad_strlsort	Сортировка списка строк	1.9.18
acad_truecolorcli	Запрос цвета из полной палитры, в режиме командной строки	1.10.21
acad_truecolordlg	Вызов диалогового окна с полной палитрой цветов	1.10.20
acdimenableupdate	Управление автоматическим обновлением ассоциативных размеров	1.18.15
acet-layerp-mark	Установка метки команды LAYERP (СЛОЙП)	1.18.17
acet-layerp-mode	Изменение режима команды LAYERP (СЛОЙП)	1.18.16
acet-laytrans	Трансляция слоев	1.18.18
acet-ms-to-ps	Преобразование длины из пространства модели в пространство листа	1.18.19
acet-ps-to-ms	Преобразование длины из пространства листа в пространство модели	1.18.20
action_tile	Действие, связываемое с элементом (напр., кнопкой) DCL-файла	3.4.1
add_list	Добавление в список диалогового окна	3.5.2
alert	Вызов окна с сообщением	1.11.12
alloc	Размер сегмента памяти	1.17.3
and	Логическое И	1.5.7

Таблица П1.1 *(продолжение)*

Функция	**Назначение**	**Номер раздела**
angle	Вычисление угла	1.6.6
angtof	Перевод угла в вещественное число	1.7.7
angtos	Перевод угла в строковое представление	1.7.8
append	Добавление к списку	1.9.2
apply	Применение операции к элементам списка	1.9.12
arx	Список загруженных ARX-приложений	1.17.10
arxload	Загрузка ARX-приложения	1.17.11
arxunload	Выгрузка ARX-приложений	1.17.12
ascii	Вычисление кода буквенно-цифрового символа	1.8.2
assoc	Выбор по ключевому элементу списка	1.9.11
atan	Арктангенс	1.6.11
atof	Перевод из строкового представления в вещественное	1.7.6
atoi	Перевод из строкового представления в целое	1.7.5
atom	Проверка символа	1.5.19
atoms-family	Перечень символов	1.16.16
autoarxload	Автозагрузка ARX-приложения	1.17.14
autoload	Автозагрузка	1.17.13
boole	Булева операция	1.6.18
boundp	Проверка значения символа	1.5.21
caaar, caddr...	(car (car (car ...))), (car (cdr (cdr ...)))	1.9.7
caar, cadr	(car (car ...)), (car (cdr ...))	1.9.7
car	Извлечение первого элемента списка	1.9.5
cdr	Получение остатка списка без первого элемента	1.9.6
chr	Получение буквенно-цифрового символа по его коду	1.8.1
client_data_tile	Привязка пользовательских данных	3.4.6
close	Закрытие файла	1.11.3

Таблица П1.1 *(продолжение)*

Функция	**Назначение**	**Номер раздела**
command	Передача команд и параметров в командную строку AutoCAD	1.3.1
cond	Проверка группы условий	1.5.13
cons	Создание списка или точечной пары	1.9.8
cos	Косинус	1.6.10
cvunit	Перевод в другие единицы измерения	1.7.11
defun	Определение новой функции	1.16.2
defun-q	Определение функции через список	1.16.4
defun-q-list-ref	Получение списка тела функции	1.16.5
defun-q-list-set	Передача списка в тело функции	1.16.6
dictadd	Добавление к словарю	1.15.5
dictnext	Переход на следующую запись словаря	1.15.4
dictremove	Удаление из словаря	1.15.6
dictrename	Переименование в словаре	1.15.7
dictsearch	Поиск в словаре	1.15.3
dimx_tile	Горизонтальный размер поля	3.6.6
dimy_tile	Вертикальный размер поля	3.6.7
distance	Расстояние	1.6.7
distof	Перевод строки в вещественное представление	1.7.9
done_dialog	Закрытие диалогового окна	3.2.5
end_image	Конец операции над графическим элементом	3.6.2
end_list	Конец операции над списком диалогового окна	3.5.3
entdel	Удаление (восстановление) примитива	1.12.4
entget	Получение списка с DXF-данными примитива	1.12.5
entlast	Извлечение последнего элемента рисунка	1.12.1
entmake	Создание примитива	1.12.6
entmakex	Создание примитива или неграфического объекта	1.12.7
entmod	Изменение примитива	1.12.8

Таблица П1.1 *(продолжение)*

Функция	**Назначение**	**Номер раздела**
entnext	Переход к следующему примитиву (подпримитиву)	1.12.2
entsel	Выбор объекта с указываемой точкой	1.12.3
entupd	Перерисовка измененного примитива	1.12.9
eq	Равенство объектов	1.5.22
equal	Равенство с допуском	1.5.23
error	Обработка ошибок	1.16.11
eval	Вычисление списка как функции	1.9.16
exit	Выход	1.16.9
exp	Вычисление e в степени x	1.6.12
expand	Расширение памяти	1.17.4
expt	Возведение в степень	1.6.13
fill_image	Заполнение графического элемента	3.6.3
findfile	Поиск файла	1.11.1
fix	Усечение вещественного числа до целого	1.6.1
float	Преобразование числа в вещественное	1.7.2
foreach	Применение операции к элементам списка	1.9.15
function	Определение функции с возможностями оптимизации	1.16.7
gc	Сборка мусора	1.17.5
gcd	Наибольший общий делитель	1.6.5
get_attr	Значение атрибута, заданное элементу диалогового окна в DCL-файле	3.4.5
get_tile	Текущее значение элемента диалогового окна	3.4.4
getangle	Ввод угла	1.10.6
getcfg	Извлечение параметров из CFG-файла	1.10.15
getcname	Получение синонимов имен команд	1.10.17
getcorner	Ввод точки с рамкой	1.10.9
getdist	Ввод расстояния	1.10.5
getenv	Значение переменной окружения	1.10.13
getfiled	Вызов диалогового окна поиска файла	1.10.18

Таблица П1.1 *(продолжение)*

Функция	**Назначение**	**Номер раздела**
getint	Ввод целого числа	1.10.3
getkword	Ввод опции из списка	1.10.2
getorient	Ввод угла	1.10.7
getpoint	Ввод точки	1.10.8
getreal	Ввод вещественного числа	1.10.4
getstring	Ввод строки	1.10.10
getvar	Значение системной переменной	1.10.11
graphscr	Переход в графический экран	1.18.3
grclear	Очистка видового экрана	1.18.9
grdraw	Рисование временного вектора	1.18.7
grread	Чтение данных через произвольное устройство ввода	1.18.10
grtext	Запись в бокс экранного меню или в строку состояния	1.18.6
grvecs	Рисование множества временных векторов	1.18.8
handent	Имя примитива, соответствующее метке	1.12.10
help	Вызов раздела справки	5.1.3
if	Условный оператор	1.5.11
initdia	Управление диалоговым окном следующей команды	1.18.11
initget	Установка опций ближайшей функции ввода	1.10.1
inters	Пересечение отрезков	1.6.19
itoa	Преобразование целого числа в строковое представление	1.7.3
lambda	Описание локальной безымянной функции	1.16.1
last	Извлечение последнего элемента списка	1.9.19
layoutlist	Перечень имен вкладок листов файла рисунка	1.15.8
length	Длина списка	1.9.21
list	Формирование списка	1.9.1
listp	Проверка на список	1.5.17

Таблица П1.1 *(продолжение)*

Функция	**Назначение**	**Номер раздела**
load	Загрузка файла с функциями	1.2.4
load_dialog	Загрузка DCL-файла	3.2.1
log	Натуральный логарифм	1.6.14
logand	Побитовое логическое И	1.6.15
logior	Побитовое логическое ИЛИ	1.6.16
lsh	Побитовый сдвиг	1.6.17
mapcar	Применение функции к каждому элементу списка	1.9.13
max	Максимум	1.6.3
mem	Параметры памяти	1.17.2
member	Проверка на вхождение в список	1.9.10
menucmd	Операция с меню или с пунктом меню	4.3.2
menugroup	Операция с группой меню	4.3.1
min	Минимум	1.6.4
minusp	Проверка на отрицательность	1.5.14
mode_tile	Переключение режима элемента диалогового окна	3.4.2
namedobjdict	Корневой словарь рисунка	1.15.2
nentsel	Доступ к данным сложного объекта с запросом	1.12.11
nentselp	Доступ к данным сложного объекта без запроса	1.12.12
new_dialog	Вызов диалогового окна	3.2.3
not	Логическое отрицание	1.5.9
nth	Выбор элемента списка по номеру	1.9.3
null	Проверка на nil	1.5.10
numberp	Проверка на число	1.5.16
open	Открытие файла	1.11.2
or	Логическое ИЛИ	1.5.8
osnap	Изменение точки с помощью объектной привязки	1.18.1

Таблица П1.1 *(продолжение)*

Функция	**Назначение**	**Номер раздела**
polar	Вычисление точки по расстоянию и направлению, задаваемому углом	1.6.20
prin1	Вывод	1.11.7
princ	Вывод	1.11.6
print	Вывод	1.11.8
progn	Программная скобка для функции if	1.5.12
prompt	Сообщение	1.11.11
quit	Выход	1.16.10
quote	Цитирование (обращение к списку не как к функции)	1.9.17
read	Чтение из строки	1.8.3
read-char	Чтение символа с клавиатуры	1.11.9
read-line	Чтение строки из файла	1.11.4
redraw	Перерисовка примитивов с возможной подсветкой	1.18.2
regapp	Регистрация приложения для расширенных данных	1.14.2
rem	Остаток от деления	1.6.2
repeat	Цикл с заданным количеством повторений	1.5.25
reverse	Переворот списка	1.9.4
rtos	Преобразование вещественного числа в строковое представление	1.7.4
set	Присвоение значения через адрес символа	1.6.24
set_tile	Задание значения переменной диалогового окна	3.4.3
setcfg	Запись в CFG-файл	1.10.16
setenv	Задание переменной окружения	1.10.14
setfunhelp	Регистрация справочной команды	5.4.1
setq	Присвоение значений символам LISP	1.1.2
setvar	Присвоение значения системной переменной AutoCAD	1.10.12
setview	Установка вида в видовом экране	1.13.6
sin	Синус	1.6.9

Таблица П1.1 *(продолжение)*

Функция	Назначение	Номер раздела
slide_image	Заполнение графического элемента слайдом	3.6.4
snvalid	Проверка имени таблицы символов	1.13.7
sqrt	Квадратный корень	1.6.8
ssadd	Добавление объекта в набор выбора	1.12.15
ssdel	Удаление объекта из набора	1.12.16
ssget	Формирование набора по запросу или признаку	1.12.14
ssgetfirst	Определение выбранных объектов	1.12.22
sslength	Количество элементов в наборе	1.12.17
ssmemb	Проверка на принадлежность объекта к набору	1.12.18
ssname	Извлечение объекта из набора	1.12.19
ssnamex	Информация о способе создания набора	1.12.20
sssetfirst	Включение ручек у объектов	1.12.21
startapp	Запуск приложения	1.17.34
start_dialog	Запуск диалогового окна загруженного DCL-файла	3.2.4
start_image	Начало операции с графическим элементом	3.6.1
start_list	Начало операции над списком диалогового окна	3.5.1
strcase	Преобразование регистра в строке символов	1.8.4
strcat	Конкатенация (сцепление) строк	1.8.5
strlen	Длина строки	1.8.6
subst	Замена элемента в списке	1.9.20
substr	Извлечение подстроки	1.8.7
tablet	Работа с планшетом	1.18.12
tblnext	Переход к следующему символу в таблице	1.13.3
tblobjname	Поиск объекта в таблице символов	1.13.4
tblsearch	Поиск символа в таблице	1.13.2

Таблица П1.1 *(продолжение)*

Функция	**Назначение**	**Номер раздела**
term_dialog	Принудительное завершение (закрытие) диалогового окна	3.2.6
terpri	Перевод курсора на новую строку	1.11.13
textbox	Вычисление ограничивающего прямоугольника для надписи	1.6.21
textpage	Переключение в текстовый экран	1.18.5
textscr	Переход в текстовое окно	1.18.4
trace	Начало трассировки	1.16.17
trans	Преобразование точки в другую систему координат	1.7.10
type	Определение типа символа	1.7.1
unload_dialog	Выгрузка DCL-файла	3.2.2
untrace	Окончание трассировки	1.16.18
vector_image	Рисование отрезка внутри графического элемента	3.6.5
ver	Определение версии AutoLISP	1.17.1
vl-acad-defun	Переопределение функции для ARX-приложений	1.16.12
vl-acad-undefun	Отмена переопределения функции	1.16.13
vl-arx-import	Импорт функции ARX-приложения в именное пространство VLX-приложения	1.17.15
vl-bb-ref	Возврат значения переменной из внедокументного именного пространства	1.17.26
vl-bb-set	Задание значения переменной из внедокументного именного пространства	1.17.25
vl-catch-all-apply	Выполнение функции с каждым элементом списка	1.9.32
vl-catch-all-error-message	Возврат сообщения об ошибке	1.9.33
vl-catch-all-error-p	Проверка, является ли аргумент объектом с информацией об ошибке	1.9.34
vl-cmdf	Передача команд и параметров в командную строку	1.3.2

Таблица П1.1 *(продолжение)*

Функция	**Назначение**	**Номер раздела**
`vl-consp`	Проверка, является ли аргумент списком, отличным от `nil`	1.5.18
`vl-directory-files`	Получение списка имен файлов папки	1.11.14
`vl-doc-export`	Экспорт функции из пространства имен VLX-приложения в текущий документ	1.17.16
`vl-doc-import`	Импорт экспортированных функций в именное пространство VLX-приложения	1.17.17
`vl-doc-ref`	Передача значения переменной из именного пространства текущего документа	1.17.28
`vl-doc-set`	Задание значения переменной из именного пространства текущего документа	1.17.27
`vl-every`	Проверка истинности условия с первым, вторым и т. д. элементами каждого из списков-аргументов	1.9.14
`vl-exit-with-error`	Передача управления функции `*error*`	1.16.14
`vl-exit-with-value`	Возврат значения функции, которая вызвана из другого пространства имен	1.16.15
`vl-file-copy`	Копирование или присоединение содержимого одного файла к другому	1.11.15
`vl-file-delete`	Удаление файла	1.11.16
`vl-file-directory-p`	Проверка, является ли строка именем папки	1.11.17
`vl-file-rename`	Переименование файла или папки	1.11.18
`vl-file-size`	Размер файла в байтах	1.11.19
`vl-file-systime`	Дата последнего изменения файла	1.11.20
`vl-filename-base`	Выделение имени файла без пути и расширения	1.11.21
`vl-filename-directory`	Выделение пути, если он входит в имя файла	1.11.22
`vl-filename-extension`	Выделение расширения из имени файла	1.11.23
`vl-filename-mktemp`	Уникальное имя для временного файла	1.11.24
`vl-get-resource`	Извлечение содержимого текстового файла, включенного в VLX-приложение	1.17.22
`vl-list*`	Построение списка	1.9.9

Таблица П1.1 *(продолжение)*

Функция	**Назначение**	**Номер раздела**
`vl-list->string`	Построение строки по кодам входящих в нее буквенно-цифровых знаков	1.8.10
`vl-list-exported-functions`	Список экспортированных функций	1.17.18
`vl-list-length`	Длина списка	1.9.22
`vl-list-loaded-vlx`	Список всех связанных с данным документом VLX-приложений с собственными именными пространствами	1.17.19
`vl-load-all`	Загрузка файла во все ранее открытые документы и во все документы, которые будут открыты в данном сеансе AutoCAD	1.17.29
`vl-load-com`	Загрузка дополнительных функций Visual LISP	6.1.2
`vl-load-reactors`	Загрузка функций, поддерживающих реакторы	9.2
`vl-member-if`	Проверка истинности условия хотя бы с одним элементом списка	1.9.23
`vl-member-if-not`	Проверка на `nil` условия хотя бы с одним элементом списка	1.9.24
`vl-mkdir`	Создание папки	1.11.25
`vl-position`	Номер объекта как элемента списка	1.9.25
`vl-prin1-to-string`	Вывод данных в строку аналогично функции `prin1`	1.11.27
`vl-princ-to-string`	Вывод данных в строку аналогично функции `princ`	1.11.26
`vl-propagate`	Передача значения переменной во все ранее открытые документы и во все документы, которые будут открыты в данном сеансе AutoCAD	1.17.30
`vl-registry-delete`	Удаление значения из реестра Windows	1.17.8
`vl-registry-descendents`	Список подразделов или параметров из указанного раздела реестра Windows	1.17.9
`vl-registry-read`	Чтение данных из раздела реестра Windows	1.17.6
`vl-registry-write`	Запись параметра раздела в реестре Windows	1.17.7
`vl-remove`	Удаление элемента из списка	1.9.26

Таблица П1.1 *(продолжение)*

Функция	**Назначение**	**Номер раздела**
vl-remove-if	Удаление элементов списка, удовлетворяющих проверяемому условию	1.9.27
vl-remove-if-not	Удаление элементов списка, не удовлетворяющих проверяемому условию	1.9.28
vl-some	Проверка выполнения условия, применяемого к последовательно передаваемым элементам списков-аргументов	1.9.29
vl-sort	Сортировка элементов списка	1.9.30
vl-sort-i	Сортировка элементов списка с возвратом номеров в новом списке	1.9.31
vl-string->list	Преобразование строки буквенно-цифровых знаков в список с кодами знаков	1.8.9
vl-string-elt	Получение кода по действующей таблице для символа с номером в строке	1.8.11
vl-string-left-trim	Удаление символов из начала строки	1.8.12
vl-string-mismatch	Вычисление длины общего префикса для двух строк, начиная с некоторой позиции	1.8.15
vl-string-position	Поиск символа с заданным кодом в строке	1.8.16
vl-string-right-trim	Удаление символов из конца строки	1.8.13
vl-string-search	Поиск заданного образца внутри строки	1.8.17
vl-string-subst	Замена внутри строки одного шаблона на другой	1.8.18
vl-string-translate	Замена символов в строке по закону	1.8.19
vl-string-trim	Удаление символов из начала и конца строки	1.8.14
vl-symbol-name	Возврат имени символа как строки	1.8.20
vl-symbol-value	Возврат значения символа	1.6.23
vl-symbolp	Проверка, является ли объект символом	1.5.20
vl-unload-vlx	Выгрузка VLX-приложения, имеющего свое именное пространство	1.17.21
vl-vbaload	Загрузка проекта VBA	1.17.23
vl-vbarun	Выполнение VBA-макроса	1.17.24
vl-vlx-loaded-p	Проверка, загружено ли VLX-приложение	1.17.20
vla-get-ActiveDocument	Указатель активного документа	6.1.5

Таблица П1.1 *(продолжение)*

Функция	**Назначение**	**Номер раздела**
vla-get-ModelSpace	Указатель пространства модели	6.1.6
vla-get-PaperSpace	Указатель пространства листа	6.1.7
vlax-3D-point	Создание точки-варианта	6.1.19
vlax-add-cmd	Добавление команды AutoCAD	6.2.1
vlax-create-object	Создание объекта внешнего приложения	6.3.8
vlax-curve-getArea	Площадь кривой	6.7.1
vlax-curve-getClosestPointTo	Вычисление точки, лежащей на кривой и ближайшей к заданной точке	6.7.15
vlax-curve-getClosestPointToProjection	Вычисление ближайшей точки на кривой после проектирования кривой на плоскость	6.7.16
vlax-curve-getDistAtParam	Длина кривой от начальной точки до точки, заданной параметром	6.7.11
vlax-curve-getDistAtPoint	Длина кривой от начальной до указанной точки	6.7.5
vlax-curve-getEndParam	Значение параметра конечной точки кривой	6.7.7
vlax-curve-getEndPoint	Вычисление конечной точки кривой	6.7.3
vlax-curve-getFirstDeriv	Вычисление первой производной в заданном месте кривой	6.7.17
vlax-curve-getParamAtDist	Значение параметра на заданном расстоянии от начальной точки кривой	6.7.10
vlax-curve-getParamAtPoint	Значение параметра в точке кривой	6.7.8
vlax-curve-getPointAtDist	Вычисление точки, находящейся на заданном расстоянии от начальной точки кривой	6.7.4
vlax-curve-getPointAtParam	Вычисление точки на кривой по заданному значению параметра	6.7.9
vlax-curve-getSecondDeriv	Вычисление второй производной в заданном месте кривой	6.7.18
vlax-curve-getStartParam	Начальное значение параметра на кривой	6.7.6
vlax-curve-getStartPoint	Вычисление начальной точки кривой	6.7.2
vlax-curve-isClosed	Проверка замкнутости кривой	6.7.12
vlax-curve-isPeriodic	Проверка периодичности кривой как функции параметра	6.7.13
vlax-curve-isPlanar	Проверка планарности кривой	6.7.14

Таблица П1.1 *(продолжение)*

Функция	Назначение	Номер раздела
vlax-dump-object	Вывод свойств VLA-объекта, а также методов, применяемых к нему	6.1.25
vlax-ename->vla-object	Преобразование примитива в VLA-объект	6.1.11
vlax-erased-p	Проверка, удален ли VLA-объект	6.3.1
vlax-for	Вычисление выражений с каждым объектом из семейства	6.3.13
vlax-get-acad-object	Вычисление указателя на корневой элемент объектной модели AutoCAD	6.1.4
vlax-get-object	Вычисление текущего вхождения объекта внешнего приложения	6.3.9
vlax-get-or-create-object	Вычисление текущего вхождения объекта приложения или создание нового	6.3.10
vlax-get-property	Извлечение свойства VLA-объекта	6.6.1
vlax-import-type-library	Импорт информации из библиотеки типов	6.3.11
vlax-invoke-method	Вызов метода ActiveX	6.2.3
vlax-ldata-delete	Удаление LISP-данных из словаря	6.4.4
vlax-ldata-get	Извлечение LISP-данных из словаря	6.4.2
vlax-ldata-list	Вывод LISP-данных словаря	6.4.3
vlax-ldata-put	Запись LISP-данных в словарь	6.4.1
vlax-ldata-test	Проверка возможности сохранения LISP-данных	6.4.5
vlax-make-safearray	Создание безопасного массива	6.1.21
vlax-make-variant	Создание варианта	6.1.15
vlax-map-collection	Применение функции ко всем объектам семейства	6.3.12
vlax-method-applicable-p	Проверка поддержки объектом метода	6.3.2
vlax-object-released-p	Проверка, освобожден ли объект в графической базе	6.3.4
vlax-product-key	Путь к разделу AutoCAD в системном реестре Windows	6.2.4
vlax-property-available-p	Проверка наличия у объекта требуемого свойства	6.6.3
vlax-put-property	Изменение значения свойства VLA-объекта	6.6.2
vlax-read-enabled-p	Проверка читаемости объекта	6.3.6

Таблица П1.1 *(продолжение)*

Функция	**Назначение**	**Номер раздела**
vlax-release-object	Освобождение объекта	6.3.5
vlax-remove-cmd	Удаление команды или группы команд	6.2.2
vlax-safearray-fill	Сохранение данных в массиве	6.1.23
vlax-safearray-get-dim	Получение размерности массива	6.5.2
vlax-safearray-get-element	Чтение элемента массива	6.5.4
vlax-safearray-get-l-bound	Нижняя граница массива	6.5.2
vlax-safearray-get-u-bound	Верхняя граница массива	6.5.3
vlax-safearray-put-element	Присвоение значения элементу массива	6.1.22
vlax-safearray-type	Определение типа данных массива	6.5.5
vlax-safearray->list	Вывод массива в список	6.5.6
vlax-tmatrix	Представление матрицы преобразования типа 4×4, используемой в операциях ActiveX	6.5.7
vlax-typeinfo-available-p	Проверка наличия информации из библиотеки типов	6.3.3
vlax-variant-change-type	Изменение типа варианта	6.1.18
vlax-variant-type	Тип данных варианта	6.1.16
vlax-variant-value	Значение варианта	6.1.17
vlax-vla-object->ename	Преобразование VLA-объекта в примитив	6.1.12
vlax-write-enabled-p	Проверка возможности изменения объекта	6.3.7
vlisp-compile	Компиляция LSP-файла в FAS-файл	1.16.8
vlr-acdb-reactor	Создание реактора операций с базой рисунка	9.5.1
vlr-add	Восстановление отключенного реактора	9.6.4
vlr-added-p	Проверка включения реактора	9.6.5
vlr-beep-reaction	Выдача звукового сигнала	9.4.2
vlr-command-reactor	Создание реактора команд	9.5.6
vlr-current-reaction-name	Имя события, вызвавшего реактор	9.6.6
vlr-data	Получение данных приложения, связанных с реактором	9.6.7
vlr-data-set	Перезапись данных приложения, связанных с реактором	9.6.8

Таблица П1.1 *(продолжение)*

Функция	Назначение	Номер раздела
vlr-deepclone-reactor	Создание реактора для операций клонирования объектов	9.5.7
vlr-docmanager-reactor	Создание реактора Диспетчера документов	9.5.2
vlr-dwg-reactor	Создание реактора для событий, связанных с сохранением, открытием, закрытием DWG-файлов	9.5.8
vlr-dxf-reactor	Создание реактора, связанного с чтением или записью DXF-файлов	9.5.9
vlr-editor-reactor	Создание реактора редактирования	9.5.5
vlr-insert-reactor	Создание реактора операций вставки блоков	9.5.10
vlr-linker-reactor	Создание реактора операций загрузки/выгрузки ARX-приложений	9.5.3
vlr-lisp-reactor	Создание реактора ввода LISP-выражений	9.5.11
vlr-miscellaneous-reactor	Создание реактора прочих событий	9.5.12
vlr-mouse-reactor	Создание реактора операций с мышью	9.5.13
vlr-notification	Проверка активности реактора в зависимости от пространства имен	9.6.12
vlr-object-reactor	Создание реактора объектов	9.5.4
vlr-owner-add	Добавление объекта к списку владельцев реактора объектов	9.6.9
vlr-owner-remove	Удаление объекта из списка владельцев реактора объектов	9.6.10
vlr-owners	Получение списка владельцев реактора объектов	9.6.11
vlr-pers	Перевод реактора в постоянный режим	9.7.1
vlr-pers-list	Получение списка постоянных реакторов	9.7.4
vlr-pers-p	Проверка режима постоянности реактора	9.7.3
vlr-pers-release	Перевод реактора во временный режим	9.7.2
vlr-reaction-names	Список всех возможных событий для данного типа реакторов	9.4.1
vlr-reaction-set	Добавление функций к реактору	9.5.20
vlr-reactions	Список точечных пар событий и действий	9.6.1
vlr-reactors	Список существующих реакторов	9.3.2

Таблица П1.1 *(окончание)*

Функция	**Назначение**	**Номер раздела**
`vlr-remove`	Отключение реактора	9.6.2
`vlr-remove-all`	Отключение реакторов определенного типа	9.6.3
`vlr-set-notification`	Задание режима работы реактора, если его именное пространство не активно	9.6.13
`vlr-sysvar-reactor`	Создание реактора, связанного с изменением значения системной переменной	9.5.14
`vlr-toolbar-reactor`	Создание реактора, связанного с изменением пиктограмм кнопок панелей инструментов	9.5.15
`vlr-trace-reaction`	Вывод аргументов реактора в окно **Trace** (Трассировка)	9.4.3
`vlr-type`	Тип реактора	9.3.3
`vlr-types`	Список всех типов реакторов	9.3.1
`vlr-undo-reactor`	Создание реактора, связанного с отменой действий	9.5.16
`vlr-wblock-reactor`	Создание реактора, связанного с командой WBLOCK (ПБЛОК)	9.5.17
`vlr-window-reactor`	Создание реактора, связанного с изменением размеров окон	9.5.18
`vlr-xref-reactor`	Создание реактора, связанного с операциями над внешними ссылками	9.5.19
`vports`	Параметры текущей конфигурации видовых экранов	1.6.22
`wcmatch`	Сравнение строки с образцом	1.8.8
`while`	Цикл по условию	1.5.24
`write-char`	Вывод символа	1.11.10
`write-line`	Вывод строки	1.11.5
`xdroom`	Определение размера свободной памяти зоны расширенных данных	1.14.5
`xdsize`	Определение размера памяти, необходимого для добавления расширенных данных	1.14.4
`zerop`	Проверка на нуль	1.5.15

ПРИЛОЖЕНИЕ 2

DXF-коды объектов AutoCAD

Структура данных основных объектов системы AutoCAD (графических примитивов и неграфических элементов) изложена в настоящем приложении в соответствии с их DXF-кодами.

В *разд. 1.12* рассмотрены функции, работающие с объектами AutoCAD и их DXF-кодами, используемыми в точечных парах спискового представления.

Данные любого графического объекта (примитива) могут быть получены с помощью LISP-функции `entget`, которая возвращает список из точечных пар и подсписков. Первый элемент каждой точечной пары является целым числом (DXF-кодом), а второй — данным этого кода. У подсписков, которые используются для хранения координат точек, первый элемент — тоже целое число, а за ним следуют вещественные числа. Для примера постройте в новом рисунке отрезок с координатами (111, 111) и (222, 222) и выполните LISP-выражение `(entget (entlast))`. Вы получите примерно такой результат:

```
((-1 . <Entity name: 7ef8fd68>) (0 . "LINE") (330 . <Entity name: 7ef8fd08>)
(5 . "2D") (100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "0") (100 .
"AcDbLine") (10 111.0 111.0 0.0) (11 222.0 222.0 0.0) (210 0.0 0.0 1.0))
```

В результирующем списке мы видим 12 элементов, из них 10 — это точечные пары с разделителем в виде точки посередине, а два элемента — списки (подсписки) из четырех числовых элементов. Каждый элемент несет свою информацию об объекте.

Общие DXF-коды

В табл. П2.1 и П2.2 приведены типы значений и расшифровка данных для кодов, общих для всех объектов. Имеются небольшие отличия указанных кодов, получаемых с помощью функций LISP и ObjectARX, от данных, фигурирующих в файлах формата DXF (сохранить рисунок в формате DXF можно с помощью

команды SAVEAS (СОХРАНИТЬКАК)). Поэтому описания, характерные только для формата DXF, помечены в таблицах признаком "DXF". Описания, действующие только при работе с приложениями LISP и ObjectARX, помечены признаком "APP".

Таблица П2.1. *Общие типы значений данных для DXF-кодов всех объектов*

Код	Тип значения данных
0—9	Строковое (зависит от реализации, обычно до 2000 символов длиной)
10—39	Трехмерная точка (вещественные координаты с двойной точностью)
40—59	Вещественное с двойной точностью
60—79	16-битовое целое
90—99	32-битовое целое
100	Строковое (до 255 символов длиной; уменьшается при использовании юникодов)
102	Строковое (до 255 символов длиной; уменьшается при использовании юникодов)
105	Строковое представление для шестнадцатеричной метки (handle)
110—149	Вещественное с двойной точностью
170—179	16-битовое целое
210—239	Вещественное с двойной точностью
270—289	16-битовое целое
290—299	Битовые флаги
300—309	Строковое
310—319	Строковая форма для шестнадцатеричного представления двоичных объектов
320—329	Строковое представление для шестнадцатеричной метки
330—369	Строковое представление для шестнадцатеричных идентификаторов (ID) объектов
370—389	16-битовое целое
390—399	Строковое представление для шестнадцатеричной метки
400—409	16-битовое целое
410—419	Строковое
420—429	32-битовое целое
430—439	Строковое
440—449	32-битовое целое
450—459	Длинное целое

Таблица П2.1 *(окончание)*

Код	Тип значения данных
460—469	Вещественное с двойной точностью
470—479	Строковое
999	Строковый комментарий
1000—1009	Строковое (ограничения такие же, как для DXF-кодов 0—9)
1010—1059	Вещественное с двойной точностью
1060—1070	16-битовое целое
1071	32-битовое целое

Таблица П2.2. *Расшифровка данных для общих DXF-кодов графических объектов*

Код	Описание
–5	APP: цепочка постоянных реакторов
–4	APP: условный оператор (используется только функцией `ssget`)
–3	APP: признак расширенных данных (фиксирован)
–2	APP: ссылочное имя примитива (фиксировано)
–1	APP: имя примитива. Постоянно только в текущем сеансе AutoCAD (в следующем сеансе изменится). Не сохраняется (фиксировано)
0	Тип примитива (фиксирован)
1	Первичное текстовое значение для примитива
2	Имя (атрибута, блока и т. п.)
3—4	Вторичное текстовое значение или имя
5	Метка примитива (строка до 16 символов длиной, фиксирована)
6	Имя типа линий (фиксировано)
7	Имя текстового стиля (фиксировано)
8	Имя слоя (фиксировано)
9	DXF: имя системной переменной (используется только в секции HEADER DXF-файла)
10	Данные первой точки примитива (начало отрезка, центр окружности и т. п.). DXF: *X*-координата первой точки (координаты *Y* и *Z* идут под кодами 20 и 30). APP: трехмерная точка (список из трех вещественных чисел)
11—18	Данные других точек. DXF: *X*-координаты других точек (координаты *Y* идут под кодами 21—28, координаты *Z* — под кодами 31—38). APP: трехмерные точки (списки из трех вещественных чисел)

Таблица П2.2 *(продолжение)*

Код	Описание
20, 30	DXF: координаты *Y* и *Z* для первой точки (см. код 10)
21—28	DXF: координаты *Y* для других точек (см. коды 11—18)
31—37	DXF: координаты *Z* для других точек (см. коды 11—18)
38	DXF: уровень примитива (если он отличен от нуля)
39	Высота (thickness) объекта, если она ненулевая (фиксирована)
40—47	Вещественные числа (высота текста, масштабные множители и т. п.)
48	Собственный масштаб типа линий (вещественное число)
49	Повторяющиеся вещественные значения (количество значений с кодом 49 задается в двузначном коде, начинающемся с цифры 7). Повторяющиеся коды группы 49 используются в таблицах переменной длины (например, длины штрихов в таблице типов линий — LTYPE)
50—58	Углы (в градусах в DXF-файлах; в радианах при чтении с помощью приложений LISP и ObjectARX)
60	Состояние видимости примитива, целое число (1 — примитив невидим; 0 или отсутствие группы 60 означает видимость)
62	Номер цвета, если не используется палитра 24-битовых цветов (фиксирован)
66	Флаг "далее следуют примитивы" (фиксирован)
67	Тип пространства объекта (модели или листа, фиксирован)
68	APP: признак, включен ли видовой экран, но целиком вне текущего окна графического экрана; или неактивен, или выключен
69	APP: номер видового экрана
70—78	Целые значения, используемые как счетчики повторений, битовые флаги, режимы
90—99	32-битовые целые числа
100	Маркер данных подкласса (со строковым именем родительского класса). Требуется для всех классов примитивов и неграфических объектов, производных от других классов. Используется для разделения данных, когда объект наследует данные от цепочки классов в ObjectARX
102	Контрольная строка, "{<*имя*>" или "}". Аналогична коду 1002 расширенных данных. После символа "{" может следовать любое имя, интерпретация которого зависит от приложения. Группа данных, предваряемая открывающей фигурной скобкой, должна затем обязательно закончиться контрольной строкой "}". Сама система AutoCAD непосредственно такие данные не использует (кроме операций проверки с помощью команды AUDIT (ПРОВЕРИТЬ)), их обработка возлагается на порождающее приложение пользователя
105	Метка объекта для входа в таблицу DIMVAR (размерных переменных)

Таблица П2.2 *(продолжение)*

Код	Описание
110	Начало ПСК (используется, если значение кода 72 равно 1). DXF: координата *X*. APP: координаты трехмерной точки
111	Вектор направления оси *X* ПСК (используется, если значение кода 72 равно 1). DXF: координата *X*. APP: координаты трехмерного вектора
112	Вектор направления оси *Y* ПСК (используется, если значение кода 72 равно 1). DXF: координата *X*. APP: координаты трехмерного вектора
120, 130	DXF: координаты *Y* и *Z* для точки начала ПСК (см. код 110)
121, 131	DXF: координаты *Y* и *Z* для вектора направления оси *X* ПСК (см. код 111)
122, 132	DXF: координаты *Y* и *Z* для вектора направления оси *X* ПСК (см. код 112)
140—149	Вещественные числа двойной точности (точки, уровень, настройки размерного стиля и т. п.)
170—179	16-битовые целые (например, битовые флаги)
210	Направление выдавливания (фиксировано). DXF: координата *X* вектора направления выдавливания. APP: координаты трехмерного вектора направления выдавливания
220, 230	DXF: координаты *Y* и *Z* вектора направления выдавливания
290—299	Флаги (булевы значения)
310—319	Произвольные двоичные объекты с представлением и ограничениями, аналогичными группе 1004: шестнадцатеричными строками длиной до 254 символов представляются группы двоичных данных длиной до 127 байт
320—329	Метки произвольных объектов, "как есть". Не изменяются в результате работы команд INSERT (ВСТАВИТЬ) и XREF (ССЫЛКА)
330—339	Метка мягкого указателя. Мягкие указатели должны действовать внутри того же рисунка или DXF-файла. Преобразуются командами INSERT (ВСТАВИТЬ) и XREF (ССЫЛКА)
340—349	Метка жесткого указателя. Жесткие указатели должны действовать внутри того же рисунка или DXF-файла. Преобразуются командами INSERT (ВСТАВИТЬ) и XREF (ССЫЛКА)
350—359	Метка мягкого указателя порождающего объекта. Указатели мягких родительских ссылок должны действовать внутри того же рисунка или DXF-файла. Преобразуются командами INSERT (ВСТАВИТЬ) и XREF (ССЫЛКА)
360—369	Метка жесткого указателя порождающего объекта. Указатели жестких родительских ссылок должны действовать внутри того же рисунка или DXF-файла. Преобразуются командами INSERT (ВСТАВИТЬ) и XREF (ССЫЛКА)

Таблица П2.2 *(продолжение)*

Код	Описание
370	Значение типа "перечисление" (enum). В графических примитивах — вес линии (`AcDb::LineWeight`)
371—379	Значения типа "перечисление"
380	Значение типа "перечисление". В графических примитивах — имя стиля печати (`AcDb::PlotStyleNameType`)
381—389	Значения типа "перечисление"
390	Строковое представление метки (в графических объектах — метки стиля печати как объекта рисунка; метка является, как правило, жестким указателем)
420—427	32-битовое представление 24-битового цвета из палитры точных цветов. Старший байт равен нулю, остальные три задают RGB-компоненты цвета (в условном шестнадцатеричном представлении — 0x00RRGGBB). Например, цвет "RGB:51,122,39" представляется десятичным числом 3 373 607
430—437	Строка с цветом из альбома цветов
440—447	32-битовое целое число. Если используется для точного цвета, то является значением прозрачности
999	DXF: строка комментария. Команда SAVEAS (СОХРАНИТЬКАК) не включает DXF-код 999 в выходной файл, однако пользователь может вручную добавить этот код с данными при редактировании DXF-файла как текста
1000	Строка длиной до 255 буквенно-цифровых знаков (используется в расширенных данных примитива, *см. разд. 1.14*)
1001	Имя приложения (строка длиной до 31 символа), для расширенных данных
1002	Контрольная строка расширенных данных ("{" или "}")
1003	Имя слоя, для расширенных данных
1004	Двоичные данные (длиной до 127 байт), для расширенных данных
1005	Строковое представление для метки, длиной до 16 шестнадцатеричных цифр (для расширенных данных)
1010	Точка, для расширенных данных. DXF: координата *X* (*Y* и *Z* располагаются в DXF-кодах 1020 и 1030). APP: трехмерная точка
1011	Трехмерная точка, интерпретируемая как положение в МСК (расширенные данные). DXF: координата *X* (*Y* и *Z* располагаются в DXF-кодах 1021 и 1031). APP: трехмерная точка
1012	Трехмерная точка, интерпретируемая как перемещение в МСК (расширенные данные). DXF: координата *X* (*Y* и *Z* располагаются в DXF-кодах 1022 и 1032). APP: трехмерная точка

Таблица П2.2 *(окончание)*

Код	Описание
1013	Трехмерная точка, интерпретируемая как направление в МСК (расширенные данные). DXF: координата *X* (*Y* и *Z* располагаются в DXF-кодах 1023 и 1033). APP: трехмерная точка
1020, 1030	DXF: координаты *Y* и *Z* точки, начинающейся в DXF-коде 1010 (для расширенных данных)
1021, 1031	DXF: координаты *Y* и *Z* точки, начинающейся в DXF-коде 1011 (для расширенных данных)
1022, 1032	DXF: координаты *Y* и *Z* точки, начинающейся в DXF-коде 1012 (для расширенных данных)
1023, 1033	DXF: координаты *Y* и *Z* точки, начинающейся в DXF-коде 1013 (для расширенных данных)
1040	Вещественные числа, для расширенных данных
1041	Расстояние, для расширенных данных
1042	Масштабный множитель, для расширенных данных
1070	16-битовое целое со знаком, для расширенных данных
1071	32-битовое длинное целое со знаком, для расширенных данных

Примечание

Объекты с жесткими указателями препятствуют автономному удалению из рисунка их дочерних объектов, а объекты с мягкими указателями — нет.

Файлы формата DXF делятся на секции со следующими именами:

- HEADER — содержит общую информацию о рисунке (номер версии и значения системных переменных);
- CLASSES — содержит информацию о классах, порожденных AutoCAD и приложениями и ссылки на которые встречаются в секциях BLOCKS, ENTITIES и OBJECTS;
- TABLES — содержит информацию о символьных таблицах (APPID, BLOCK_RECORD, DIMSTYLE, LAYER, LTYPE, STYLE, UCS, VIEW, VPORT);
- BLOCKS — содержит информацию об описаниях блоков и элементах блоков;
- ENTITIES — содержит информацию о примитивах;
- OBJECTS — содержит информацию о неграфических объектах, не относящихся к таблицам;
- THUMBNAILIMAGE — содержит растровый образец для быстрого просмотра рисунка (данная секция может отсутствовать).

Наибольший интерес для нас представляют данные, хранящиеся в секции ENTITIES. В табл. П2.3 приводятся типы данных с DXF-кодами, которые являются общими для всех видов графических примитивов.

Если ваша программа будет создавать в рисунке новые примитивы, то, формируя список с данными этих объектов, вам следует заполнять только те DXF-коды, значения которых отличны от значений по умолчанию, поскольку значения по умолчанию система AutoCAD добавит самостоятельно.

Таблица П2.3. *Данные общих DXF-кодов, используемых в секции ENTITIES*

Код	Значение	По умолчанию
–1	APP: имя примитива (изменяется в каждом сеансе редактирования рисунка)	Нет
0	Тип примитива	Нет
5	Метка	Нет
6	Имя типа линий	"BYLAYER" ("ПОСЛОЮ")
8	Имя слоя	Нет
48	Собственный масштаб типа линий	1.0
60	Видимость объекта (0 — видимый, 1 — невидимый)	0
62	Номер цвета; 0 соответствует значению "BYBLOCK" ("ПОБЛОКУ"). Отрицательное значение для цвета слоя означает, что слой выключен	"BYLAYER" ("ПОСЛОЮ")
67	1 — пространство листа, 0 — пространство модели	0
92	Размер растрового образца объекта-заместителя в байтах (см. код 310)	Нет
100	Маркер подкласса (имеет вид: "AcDbEntity" и "AcDb" + тип примитива)	Нет
102	Начало группы данных приложения (имеет вид "{<*имя*>", причем <*имя*> может быть опущено) или конец группы данных приложения (имеет вид "}"). Строка "{ACAD_REACTORS" означает начало данных группы постоянных реакторов. Строка "{ACAD_XDICTIONARY" указывает начало словаря расширения объекта	Нет
310	Данные объекта-заместителя (может быть несколько строк, не более 256 символов в строке)	Нет
330	Мягкий указатель или метка владельца (словаря), которому принадлежит объект	Нет
360	Жесткий указатель или метка владельца (словаря), которому принадлежит объект	Нет
370	Вес, преобразованный в целое число умножением на 100	Нет

Таблица П2.3 *(окончание)*

Код	Значение	По умолчанию
410	APP: имя вкладки	Нет
420	Значение точного (24-битового) цвета из полной палитры	Нет
430	Имя из альбома цветов	Нет
440	Прозрачность	Нет

В рисунке встречаются примитивы следующих типов: 3DFACE, 3DSOLID, ACAD_PROXY_ENTITY, ACAD_TABLE, ARC, ATTDEF, ATTRIB, BODY, CIRCLE, DIMENSION, ELLIPSE, HATCH, IMAGE, INSERT, LEADER, LINE, LWPOLYLINE, MLINE, MTEXT, OLEFRAME, OLE2FRAME, POINT, POLYLINE, RAY, REGION, SEQEND, SHAPE, SOLID, SPLINE, TEXT, TOLERANCE, TRACE, VERTEX, VIEWPORT, WIPEOUT, XLINE. Общие для этих примитивов DXF-коды рассмотрены в табл. П2.3.

Конкретные DXF-коды для основных примитивов рассмотрены в *разд. "Некоторые примитивы"* данного приложения. Остальные можно найти в справочной системе AutoCAD или в книге "AutoCAD 2005" (серия "В подлиннике")[1].

Символьные таблицы

В каждом рисунке находятся символьные таблицы, которые хранят информацию о некоторых именованных неграфических объектах (слоях, блоках и т. д.). Таблицы располагаются в DXF-формате в секции TABLES.

Таких таблиц девять:

- APPID — зарегистрированные приложения;
- BLOCK_RECORD — блоки;
- DIMSTYLE — размерные стили;
- LAYER — слои;
- LTYPE — типы линий;
- STYLE — текстовые стили;
- UCS — системы координат;
- VIEW — виды;
- VPORT — конфигурации видовых экранов.

Работа с символьными таблицами ведется с помощью функций `tblnext`, `tblsearch` и `tblobjname`, которые возвращают список с данными о неграфическом объекте.

[1] Полещук Н. Н. AutoCAD 2005. — СПб.: БХВ-Петербург, 2005. — 944 с.

Организация этого списка аналогична организации списка, возвращаемого функцией `entget` для графического объекта. В табл. П2.4 приведена расшифровка DXF-кодов для объектов символьных таблиц. Некоторые из приведенных DXF-кодов в реальном объекте могут отсутствовать.

Таблица П2.4. *Расшифровка данных для общих DXF-кодов объектов символьных таблиц*

Код	Описание
–1	APP: имя примитива. Постоянно только в текущем сеансе AutoCAD (в следующем сеансе изменится). Не сохраняется
0	Тип примитива (имя таблицы, фиксирован)
2	Имя объекта таблицы
5	Метка примитива (строка длиной до 16 символов, недоступна для изменения). Для всех таблиц, кроме DIMSTYLE
105	Метка примитива (строка до 16 символов длиной, недоступна для изменения). Только в таблице DIMSTYLE
102	Признак начала данных приложения, имеет вид "{*имя*". Например, "{ACAD_REACTORS" является признаком начала данных постоянных реакторов
Коды с данными приложения	Данные, сформированные приложением (располагаются между двумя кодами 102, играющими роль открывающей и закрывающей скобок)
330	Мягкий указатель или метка владельца-словаря
102	Конец данных постоянного реактора, имеет вид "}"
102	"{ACAD_XDICTIONARY" — начало данных словаря, привязанного к объекту таблицы (extension dictionary)
360	Жесткий указатель или метка владельца-словаря
102	Конец данных словаря, имеет вид "}"
330	Мягкий указатель или метка владельца-объекта
100	Маркер подкласса ("AcDbSymbolTableRecord")
70	Максимальное количество элементов таблицы

Конкретные DXF-коды для объектов символьных таблиц приведены в справочной системе AutoCAD или в книге "AutoCAD 2005" (серия "В подлиннике")[2].

[2] Полещук Н. Н. AutoCAD 2005. — СПб.: БХВ-Петербург, 2005. — 944 с.

Некоторые примитивы

В табл. П2.5—П2.11 приведены описания DXF-кодов для основных примитивов: LINE, CIRCLE, LWPOLYLINE, POLYLINE, VERTEX, SEQEND, INSERT. Общие для этих примитивов DXF-коды рассмотрены в табл. П2.3.

Таблица П2.5. *Данные DXF-кодов, используемые в секции ENTITIES примитива LINE*

Код	Значения данных	По умолчанию
100	Маркер подкласса ("AcDbLine")	Нет
39	Высота (величина выдавливания)	0
10	Начальная точка (в МСК). DXF: координата *X* начальной точки (координаты *Y* и *Z* идут под кодами 20 и 30). APP: трехмерная точка	Нет
20	DXF: координата *Y* начальной точки	Нет
30	DXF: координата *Z* начальной точки	Нет
11	Конечная точка (в МСК). DXF: координата *X* конечной точки (координаты *Y* и *Z* идут под кодами 21 и 31). APP: трехмерная точка	Нет
21	DXF: координата *Y* конечной точки	Нет
31	DXF: координата *Z* конечной точки	Нет
210	Вектор направления выдавливания (начинается в точке (0,0,0)). DXF: координата *X* вектора выдавливания (координаты *Y* и *Z* идут под кодами 220 и 230). APP: трехмерная точка	DXF: 0 APP: (0, 0, 1)
220	DXF: координата *Y* вектора выдавливания	0
230	DXF: координата *Z* вектора выдавливания	1

Таблица П2.6. *Данные DXF-кодов, используемые в секции ENTITIES примитива CIRCLE*

Код	Значения данных	По умолчанию
100	Маркер подкласса ("AcDbCircle")	Нет
39	Высота (величина выдавливания)	0
10	Точка центра (в СК объекта). DXF: координата *X* точки центра (координаты *Y* и *Z* идут под кодами 20 и 30). APP: трехмерная точка	Нет

Таблица П2.6 *(окончание)*

Код	Значения данных	По умолчанию
20	DXF: координата *Y* точки центра	Нет
30	DXF: координата *Z* точки центра	Нет
40	Радиус	Нет
210	Вектор направления выдавливания (начинается в точке (0,0,0)). DXF: координата *X* вектора выдавливания (координаты *Y* и *Z* идут под кодами 220 и 230). APP: трехмерная точка	DXF: 0 APP: (0, 0, 1)
220	DXF: координата *Y* вектора выдавливания	0
230	DXF: координата *Z* вектора выдавливания	1

Таблица П2.7. *Данные DXF-кодов, используемые в секции ENTITIES примитива LWPOLYLINE*

Код	Значения данных	По умолчанию
100	Маркер подкласса ("AcDbPolyline")	Нет
90	Количество вершин	Нет
70	Сумма включенных битовых флагов (1 — замкнутая полилиния; 128 — режим применения типа линий ко всему объекту в целом, а не к отдельным участкам)	0
43	Постоянная ширина. Не используется при переменной ширине (см. DXF-коды 40 и 41)	0
38	Уровень	0
39	Высота (величина выдавливания)	0
10	Вершина (в СК объекта). Повторяется столько раз, каково количество вершин в полилинии. DXF: координата *X* вершины (координата *Y* идет под кодом 20). APP: двумерная точка	Нет
20	DXF: координата *Y* вершины	Нет
40	Начальная ширина участка. Повторяется столько раз, каково количество вершин в полилинии. Не используется при постоянной ширине (см. DXF-код 43)	0
41	Конечная ширина участка. Повторяется столько раз, каково количество вершин в полилинии. Не используется при постоянной ширине (см. DXF-код 43)	0
42	Коэффициент кривизны (bulge) — тангенс четверти угла, стянутого дуговым сегментом. 0 — для прямолинейного сегмента, 1 — для полуокружности. К кривизне добавляется знак минус, если она идет по часовой стрелке. Повторяется столько раз, каково количество вершин в полилинии	0

Таблица П2.7 *(окончание)*

Код	Значения данных	По умолчанию
210	Вектор направления выдавливания (начинается в точке (0,0,0)). DXF: координата *X* вектора выдавливания (координаты *Y* и *Z* идут под кодами 220 и 230). APP: трехмерная точка	DXF: 0 APP: (0, 0, 1)
220	DXF: координата *Y* вектора выдавливания	0
230	DXF: координата *Z* вектора выдавливания	1

В табл. П2.8 рассматривается структура данных примитива типа POLYLINE. Используется сокращение: СКО — система координат объекта.

В базе рисунка вслед за основным примитивом POLYLINE должны обязательно идти подпримитивы (это примитивы типа VERTEX с данными о вершинах и примитив типа SEQEND, являющийся признаком конца полилинии). Структура примитивов типа VERTEX и SEQEND рассматривается в табл. П2.9 и П2.10.

Под одним именем POLYLINE фактически сохраняются четыре типа объектов: двумерная полилиния, трехмерная полилиния, полигональная сеть и многогранная сеть.

Таблица П2.8. *Данные DXF-кодов, используемые в секции ENTITIES примитива POLYLINE*

Код	Значения данных	По умолчанию
100	Маркер подкласса ("AcDb2dPolyline", "AcDb3dPolyline", "AcDbPolygonMesh" и "AcDbPolyFaceMesh")	Нет
10	DXF: 0 APP: фиктивная трехмерная точка, у которой координаты *X* и *Y* равны 0, а координата *Z* равна уровню объекта (в СКО — для двумерной полилиниии, в МСК — для трехмерной полилинии)	Нет
20	DXF: 0	Нет
30	DXF: для двумерной полилинии — уровень объекта в СКО, для трехмерной — уровень в МСК	Нет
39	Высота (величина выдавливания)	0
70	Сумма включенных битовых флагов (1 — замкнутая полилиния или полигональная сеть, замкнутая в направлении M; 2 — опцией **Fit** (СГладить) при редактировании добавлены вспомогательные вершины; 4 — опцией **Spline** (СПлайн) при редактировании добавлены вспомогательные вершины; 8 — признак трехмерной полилинии; 16 — признак полигональной сети из четырехугольных ячеек; 32 — полигональная сеть замкнута в направлении N; 64 — признак многогранной сети; 128 — режим применения типа линий ко всей полилинии в целом, а не к отдельным участкам)	0

Таблица П2.8 *(окончание)*

Код	Значения данных	По умолчанию
40	Начальная ширина полилинии	0
41	Конечная ширина полилинии	0
71	Количество вершин сети в направлении M	0
72	Количество вершин сети в направлении N	0
73	Плотность сглаживания сети в направлении M	0
74	Плотность сглаживания сети в направлении N	0
75	Тип сглаживания полилинии или сети. Одно из целочисленных значений: 0 — без сглаживания; 5 — квадратичный B-сплайн; 6 — кубический B-сплайн; 8 — поверхность Безье	0
210	Вектор направления выдавливания (начинается в точке (0,0,0)). DXF: координата *X* вектора выдавливания (координаты *Y* и *Z* идут под кодами 220 и 230). APP: трехмерная точка	DXF: 0 APP: (0, 0, 1)
220	DXF: координата *Y* вектора выдавливания	0
230	DXF: координата *Z* вектора выдавливания	1

После обычных данных полилинии в базе рисунка могут располагаться еще расширенные данные стандартного приложения с именем "AUTOCAD_POSTSCRIPT_FIGURE", обеспечивающего вывод на принтер в формате PostScript.

Замечание

Многогранная сеть тоже является разновидностью примитива POLYLINE. Ее признаком служит наличие битового флага 64 в DXF-коде 70. В этом случае DXF-код 71 задает количество вершин в сети, а DXF-код 72 — количество полиграней.

Грани многогранной сети (полиграни) могут иметь произвольное количество вершин. Многогранные сети создаются командой PFACE (ПГРАНЬ). В базе рисунка вслед за основным примитивом POLYLINE идут примитивы VERTEX, которые сначала описывают положение вершин, а затем — полиграней.

Таблица П2.9. *Данные DXF-кодов, используемые в секции ENTITIES примитива VERTEX*

Код	Значения данных	По умолчанию
100	Маркер подкласса ("AcDbVertex", "AcDb2dVertex", "AcDb3dPolylineVertex", "AcDbPolygonMeshVertex" и "AcDbPolyFaceMeshVertex")	Нет
10	Вершина (для двумерной полилинии — в СКО, для трехмерной полилинии или сети — в МСК). DXF: координата *X* вершины (координаты *Y* и *Z* идут под кодами 20 и 30). APP: двумерная или трехмерная точка	Нет

Таблица П2.9 *(окончание)*

Код	Значения данных	По умолчанию
20	DXF: координата *Y* вершины	Нет
30	DXF: координата *Z* вершины	Нет
40	Начальная ширина полилинии	0
41	Конечная ширина полилинии	0
42	Кривизна (bulge) — тангенс четверти угла, стянутого дуговым сегментом. 0 — для прямолинейного сегмента, 1 — для полуокружности. К кривизне добавляется знак минус, если она идет по часовой стрелке	0
70	Сумма включенных битовых флагов (1 — вспомогательная вершина, добавленная опцией **Fit** (СГладить) команды PEDIT (ПОЛРЕД) при редактировании; 2 — в вершине задано направление касательной для опции **Fit** (СГладить); 8 — вспомогательная вершина, добавленная опцией **Spline** (СПлайн) команды PEDIT (ПОЛРЕД); 16 — управляющая точка каркаса сплайна; 32 — точка трехмерной полилинии; 64 — точка полигональной сети; 128 — точка многогранной сети). Битовый флаг 4 не используется	0
50	Направление касательной для опции **Fit** (СГладить) команды PEDIT (ПОЛРЕД)	Нет
71	Номер вершины полиграни	0
72	Номер вершины полиграни	0
73	Номер вершины полиграни	0
74	Номер вершины полиграни	0

Если примитив VERTEX определяет вершину полиграни, то для нее включены битовые флаги 64 и 128, а DXF-коды 10, 20 и 30 задают координаты вершины.

Если примитив VERTEX определяет саму полигрань, то для нее включен битовый флаг 128 и выключен битовый флаг 64. DXF-коды 10, 20 и 30 не задают никаких координат и обычно равны нулю. Данные DXF-кодов 71—74 определяют вершины полиграни. Если в полиграни меньше четырех вершин, то лишние заполняются нулями. Знак минус перед номером вершины означает, что кромка, начинающаяся в данной вершине, имеет признак невидимости.

Если полиграни, создаваемые командой PFACE (ПГРАНЬ), имеют более четырех вершин, то AutoCAD самостоятельно разбивает полиграни на более мелкие.

Таблица П2.10. *Данные DXF-кодов, используемые в секции ENTITIES примитива SEQEND*

Код	Значения данных	По умолчанию
–2	APP: имя порождающего примитива (типа POLYLINE или INSERT)	Нет

Примитив SEQEND может завершать данные не только по вершинам полилинии, но и данные по атрибутам вхождения блока.

Примитивы INSERT, структура которых рассматривается в табл. П2.11, соответствуют вхождениям блока (т. е. объектам рисунка, полученным командой INSERT (ВСТАВИТЬ) для блоков).

Таблица П2.11. *Данные DXF-кодов, используемые в секции ENTITIES примитивов INSERT*

Код	Значения данных	По умолчанию
100	Маркер подкласса ("AcDbBlockReference")	Нет
66	Признак следования атрибутов (1 — да, 0 — нет)	0
2	Имя блока	Нет
10	Точка вставки блока (в СКО). DXF: координата *X* точки вставки (координаты *Y* и *Z* идут под кодами 20 и 30). APP: трехмерная точка	Нет
20	DXF: координата *Y* точки вставки	Нет
30	DXF: координата *Z* точки вставки	Нет
41	Масштабный коэффициент вставки по оси *X*	1
42	Масштабный коэффициент вставки по оси *Y*	1
43	Масштабный коэффициент вставки по оси *Z*	1
50	Угол поворота	0
70	Количество столбцов для массива блоков	1
71	Количество рядов для массива блоков	1
44	Расстояние между столбцами	0
45	Расстояние между рядами	0
210	Вектор направления выдавливания (начинается в точке (0,0,0)). DXF: координата *X* вектора выдавливания (координаты *Y* и *Z* идут под кодами 220 и 230). APP: трехмерная точка	DXF: 0 APP: (0, 0, 1)
220	DXF: координата *Y* вектора выдавливания	0
230	DXF: координата *Z* вектора выдавливания	1

Если примитив INSERT является вхождением блока с атрибутами (см. DXF-код 66), то тогда после самого INSERT в базе рисунка следуют примитивы ATTRIBUTE. Признаком конца атрибутов является примитив SEQEND.

Для вхождения динамического блока с помощью функции `entget` можно получить примерно такой список:

```
((-1 . <Entity name: 7ef81408>) (0 . "INSERT") (5 . "169") (102 .
"{ACAD_XDICTIONARY") (360 . <Entity name: 7ef814a0>) (102 . "}") (330 .
<Entity name: 7ef7fcf8>) (100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 .
"0") (100 . "AcDbBlockReference") (2 . "*U3") (10 0.0 2000.0 0.0) (41 . 1.0)
(42 . 1.0) (43 . 1.0) (50 . 0.0) (70 . 0) (71 . 0) (44 . 0.0) (45 . 0.0) (210
0.0 0.0 1.0))
```

Этот список отличается от списка с данными вхождения статического блока именем блока (оно начинается с символов *U, что соответствует анонимным блокам). Кроме того, примитив INSERT имеет словарь расширения, который предоставляет доступ к динамическим параметрам.

ПРИЛОЖЕНИЕ 3

Индикатор

Графические элементы DCL-окон позволяют создать имитатор индикатора длительного процесса (progress bar) — отдельное окно с бегущей полосой (рис. П3.1). Такое окно можно вызывать в момент какого-то длинного вычисления.

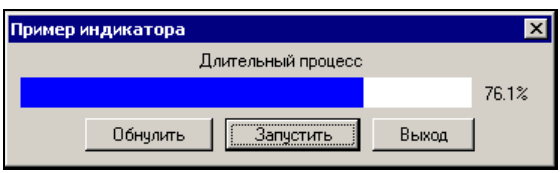


Рис. П3.1. Пример индикатора процесса

В листинге П3.1 показан текст DCL-файла, соответствующего диалоговому окну, приведенному на рисунке.

Листинг П3.1. Файл progress.dcl

```
// Приложение 3\progress.dcl
// © Н.Н.Полещук, 2006 г.
// В книге: Н.Н.Полещук, П.В.Лоскутов
//"AutoLISP и Visual LISP в среде AutoCAD"
// (издательство "БХВ-Петербург", 2006)
//

progress: dialog{label="Пример индикатора";
:text{label="Длительный процесс";alignment=centered;}
:row{
  :image{width=50;height=1.5;key="img";}
  :text{label="0%    ";key="cif";}
}//row
```

```
:row{fixed_width=true;alignment=centered;
  :button{label="Обнулить";key="0";}
  :button{label="Запустить";key="start";}
  :ok_button{label="Выход";is_cancel=true;}
}//row
};
```

Текст LISP-функций, управляющих работой диалога, дан в листинге П3.2.

Листинг П3.2. Файл pbar.lsp

```
; Приложение 3\pbar.lsp
; © Н.Н.Полещук, 2006 г.
; В книге: Н.Н.Полещук, П.В.Лоскутов
;"AutoLISP и Visual LISP в среде AutoCAD"
; (издательство "БХВ-Петербург", 2006)
;
; Функция управления диалогом
(defun pbar ( / dc dx dy)
(if (> (setq dc (load_dialog "progress.dcl")) 0)
  (if (new_dialog "progress" dc)
    (progn
      (setq dx (dimx_tile "img") dy (dimy_tile "img"))
      (annul dx dy)
      (action_tile "0" "(annul dx dy)")
      (action_tile "start" "(startpb dx dy)")
      (start_dialog)
      (done_dialog)
      (unload_dialog dc)
    );progn
    (princ "\nНе загрузить диалог progress. ")
  );if
  (princ "\nНе найден файл progress.dcl. ")
);if
(princ)
);defun pbar
; Функция запуска индикатора
(defun startpb (x y / i)
(setq i 0)
(repeat 100000
  (setq i (1+ i))
  (start_image "img")
  (fill_image 0 0 (/ (* i x) 100000) y 5)
  (end_image)
```

```
  (set_tile "cif" (strcat (rtos (/ (float i) 1000) 2 1) "%"))
);repeat
);defun startpb
; Функция обнуления индикатора
(defun annul (x y / )
  (start_image "img")
  (fill_image 0 0 x y 0)
  (end_image)
  (set_tile "cif" "0%")
);defun annul
```

Сам индикатор представляет собой прямоугольник (элемент типа `image` с кодом `"img"`), который в нулевом состоянии заполнен белым цветом. По мере выполнения цикла прямоугольник с помощью функции `fill_image` постепенно заполняется синим цветом, начиная с левой части.

Справа от графического элемента в диалоговом окне находится текстовое поле, в котором отражается текущее числовое состояние счетчика процесса, изменяющегося от 0 до 100% (с выводом одного знака после десятичной точки).

В листинге П3.2 приведены три функции: `pbar` загружает диалоговое окно и управляет им, `startpb` запускает индикатор, `annul` очищает индикатор и приводит его в начальное состояние.

Для демонстрации индикатора важно правильно подобрать количество повторений к функции `repeat`, поскольку при малом значении процесс заканчивается слишком быстро, а при большом он, наоборот, затягивается. Автор для условий своего компьютера (Pentium 4, 2.53 ГГц) выбрал значение 100 000. В ваших условиях это число может быть другим.

ПРИЛОЖЕНИЕ 4

Светофор

При проектировании приложений с диалоговыми окнами на языке DCL приходится решать проблемы эффективности и понятности таких окон для пользователей различной квалификации. В этом направлении вполне возможно использование цветовых эффектов, что достигается с помощью графических элементов диалога.

Другая проблема DCL-окон — **контроль данных**, получаемых через текстовые поля типа `edit_box`. Удобных встроенных средств, как в C++, в языке LISP нет. Более того, отсутствие необходимой проверки данных, вводимых пользователем, может вызывать неприятные прерывания при работе вашей программы.

В рассматриваемом в этом приложении примере графические элементы оформлены в виде светофора, огни которого изменяют свое состояние в зависимости от действий пользователя. На рис. П.4.1—П4.4 приведены окна с различными состояниями светофора.

В листинге П4.1 приведен текст DCL-файла, соответствующего диалоговому окну со светофором, приведенному на рисунках.

Текст LISP-функций, управляющих работой диалога-светофора, показан в листинге П4.2.

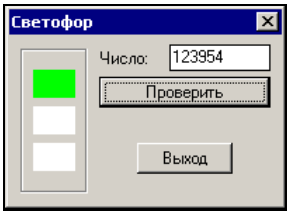


Рис. П4.1. Светофор с положительным числом

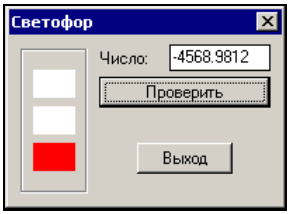


Рис. П4.2. Светофор с отрицательным числом

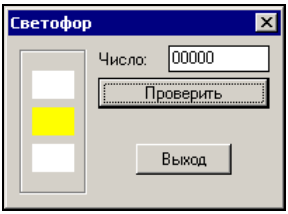


Рис. П4.3. Светофор с нулевым числом

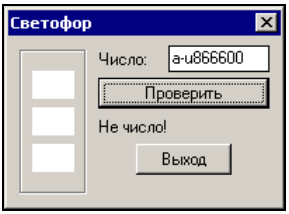


Рис. П4.4. Светофор с сообщением об ошибке

Листинг П4.1. Файл lights.dcl

```
// Приложение 4\lights.dcl
// © Н.Н.Полещук, 2006 г.
// В книге: Н.Н.Полещук, П.В.Лоскутов
//"AutoLISP и Visual LISP в среде AutoCAD"
// (издательство "БХВ-Петербург", 2006)
//

lights: dialog{label="Светофор";initial_focus="num";
:row{
  :boxed_column{
    :image{width=5;height=1.5;key="pos";fixed_width=true;}
    :image{width=5;height=1.5;key="0";fixed_width=true;}
    :image{width=5;height=1.5;key="neg";fixed_width=true;}
    :spacer{height=0.3;}
  }//row
  :column{alignment=top;fixed_height=true;
    :edit_box{label="Число:";key="num";alignment=centered;
      edit_limit=10;edit_width=10;fixed_width=true;}
    :button{label="Проверить";key="check";}
    :text{label="";key="errorn";}
    :ok_button{label="Выход";is_cancel=true;}
  }//column
}
}//lights
```

Листинг П4.2. Файл checknum.lsp

```
; Приложение 4\checknum.lsp
; © Н.Н.Полещук, 2006 г.
; В книге: Н.Н.Полещук, П.В.Лоскутов
;"AutoLISP и Visual LISP в среде AutoCAD"
; (издательство "БХВ-Петербург", 2006)
;
```

```
(defun checknum ( / dc nn)
(if (> (setq dc (load_dialog "lights.dcl")) 0)
  (if (new_dialog "lights" dc)
    (progn
      (setq nn "")
      (setq dx (dimx_tile "pos") dy (dimy_tile "pos"))
      (action_tile "num" "(setq nn $value)")
      (action_tile "check" "(chsign nn dx dy)")
      (start_dialog)
      (done_dialog)
      (unload_dialog dc)
    );progn
    (princ "\nНе загрузить диалог lights. ")
  );if
  (princ "\nНе найден файл lights.dcl. ")
);if
(princ)
);defun checknum
; Функция проверки правильности числа и анализа его знака
(defun chsign (n x y / good eo res c1 c2 c3)
  ; Отбрасываем пробелы слева и справа
  (setq good T n (vl-string-trim " " n))
  ; Проверяем на пустой ввод и внутренние пробелы
  (if (or (= n "") (vl-string-position (ascii " ") n)) (setq good nil))
  ; Проверяем, читается ли это
  (if good
    (progn
      (setq eo (vl-catch-all-apply 'read (list n)))
      (setq good (numberp eo)); nil - ошибки в числе
    );progn
  );if
  ; Проверяем, число ли введено
  (if good
    (progn
      (setq res (read n))
      (if (not (numberp res)) (setq good nil))
    );progn
  );if
  ; Установка огней светофора
  ; Или сообщение: Не число!
  (setq c1 0 c2 0 c3 0)
  (set_tile "errorn" "")
  (if good
; Три варианта огней светофора, в зависимости от знака числа
    (cond
      ((> res 0) (setq c3 3))
```

```
      ((< res 0) (setq c1 1))
      (T (setq c2 2))
    );cond
    (set_tile "errorn" "Не число!")
  );if
  (start_image "neg")
  (fill_image 0 0 x y c1)
  (end_image)
  (start_image "0")
  (fill_image 0 0 x y c2)
  (end_image)
  (start_image "pos")
  (fill_image 0 0 x y c3)
  (end_image)
);defun chsign
```

Первой функцией в листинге П4.2 является функция `checknum` — она загружает окно с диалогом `lights`. В этом окне с помощью кнопки `"check"` запускается центральная функция приложения — `chsign`, которой передаются значения трех аргументов:

- `nn` — данное, введенное пользователем в текстовом поле с кодом `"num"`;
- `dx` — горизонтальный размер огня светофора (графического элемента типа `image`);
- `dy` — вертикальный размер огня светофора (графического элемента типа `image`).

Данное в переменной `nn` является строковым и оно предположительно после чтения строки должно соответствовать числу (целому или вещественному). Мы в своем приложении вводим также ограничение: `nn` не должно быть пустым или только из пробелов (это будет интерпретироваться как ошибка — не число).

Функция `chsign` должна проверить значение `nn` на правильность (не должно быть недопустимых символов, а введенный текст должен отображать число), и если число правильное, то определить его знак и зажечь соответствующий огонь светофора.

Дальнейшая проверка значения `nn` и преобразование его в число представляет интерес для всех, кто читает данные через DCL-окно.

После передачи значения `nn` из функции `checknum` в функцию `chsign` оно оказывается в переменной `n`, являющейся аргументом функции `chsign`. Первое, что делает функция `chsign` с аргументом `n`, — отбрасывает слева и справа все пробелы (лишние крайние пробелы не видны в поле ввода, поэтому они не должны вызывать ошибки):

```
(setq n (vl-string-trim " " n))
```

Если после отбрасывания концевых пробелов в `n` осталась пустая строка — выдается признак ошибки: `(setq good nil)`. Такой же признак ошибки формируется, если внутри `n` остались пробелы (в числе не может быть внутренних пробелов):

```
(if (or (= n "") (vl-string-position (ascii " ") n)) (setq good nil))
```

Наличие внутренних пробелов определяется с помощью функции `vl-string-position`.

После того как из `n` исключили и крайние, и внутренние пробелы, осталось проанализировать, представляет ли оставшаяся группа буквенно-цифровых символов число или нет. Проще всего это сделать функцией `read` — она сразу же выдает нам и число. Однако эта функция "буксует" в некоторых ситуациях.

Попробуйте, например, выполнить в командной строке выражение с двумя подряд точками: `(read "7..9")`. Результатом будет `7`. Вроде бы и неплохо (хотя само по себе преобразование `7..9` в `7` вызывает вопросы), но самое главное и неочевидное — результат не является числом!

Выявить это можно с помощью функции `type`:

`(type (read "7..9"))` возвратит `SYM`.

Таким образом в ошибочных ситуациях мы рискуем получить не число, а символ. Вместо функции `type` можно было для проверки воспользоваться функцией `numberp`.

А попробуйте теперь с помощью функции `read` прочитать строку из одной точки: `(read ".")`. Получите прерывание с неприятным сообщением:

; error: misplaced dot on input

(; ошибка: неверно расположенная точка на входе)

Таким образом, помимо проверки типа, прочитанного с помощью `read` значения, необходимо еще уметь **обрабатывать прерывания**. Такой инструмент у нас есть — это великолепная функция `vl-catch-all-apply`.

Обратите внимание на следующее выражение из функции `chsign`:

```
(setq eo (vl-catch-all-apply 'read (list n)))
```

Оно предназначено для выполнения функции `read` со всеми элементами списка `(list n)`, т. е. только с `n`. Если ошибок нет, то `eo` будет присвоено значение, вычисленное функцией `read`, а если возникло прерывание из-за ошибки, то `eo` получит значение объекта типа VL-CATCH-ALL-APPLY-ERROR. От ошибочного объекта нам достаточно только того, что это не число и функция `numberp` для него вернет `nil`.

Поэтому выражение `(setq good (numberp eo))` присвоит признаку `good` значение `T` только тогда, когда в `n` занесено правильное число.

Указанная связка функций `read` и `vl-catch-all-apply` позволила решить довольно сложную проблему малой кровью.

В оставшейся части функции `chsign` производится анализ знака введенного в диалоговом окне числа и для положительного числа зажигается верхний зеленый огонь (см. рис. П4.1), для отрицательного — нижний красный огонь (см. рис. П4.2), а для нуля — средний желтый огонь (см. рис. П4.3).

Если же был выработан признак ошибки во введенном числе, то в поле с кодом `"errorn"` выводится сообщение `"Не число!"`.

ПРИЛОЖЕНИЕ 5

Танцы панелей

Функционирование панелей инструментов похоже на функционирование меню (например, падающих), однако работать из LISP-программы с панелями труднее, потому что приходится в обязательном порядке применять технологию ActiveX.

В данном приложении приводится пример решения в системе AutoCAD 2006 следующих задач:

- создание новой панели в оперативной памяти;
- добавление кнопок к панели;
- настройка изображения на кнопке панели;
- настройка функциональности кнопки (задание макроса и подсказок);
- перемещение панели на экране;
- удаление панели.

В рассматриваемом далее исходном коде панель сначала будет создана и оформлена. Затем эта панель будет передвигаться по экрану с заданной скоростью и по заданному маршруту. По окончании "танца" панель будет удалена. Для версий AutoCAD, отличных от 2006, алгоритм будет тот же, но в коде будут небольшие изменения (соответствующие примечания будут сделаны).

В листинге П5.1 показан текст функции `cr_tb`, предназначенной для создания новой панели в заданной группе адаптации (для версии 2005 и ниже — заданной группе меню). В процессе работы необходимо проанализировать правильность имени группы, правильность имени панели (нет ли уже с таким именем).

Листинг П5.1. Файл cr_tb.lsp

```
; Приложение 5\cr_tb.lsp
; © Н.Н.Полещук, 2006 г.
```

```
; В книге: Н.Н.Полещук, П.В.Лоскутов
;"AutoLISP и Visual LISP в среде AutoCAD"
; (издательство "БХВ-Петербург", 2006)
;
(defun cr_tb (g_name t_name l_names l_helps l_macros
  l_icons cx cy nrows / wbOK mgs mg tbs tb i tbitem)
(vl-load-com)
(setq mgs (vla-get-MenuGroups (vlax-get-acad-object)))
(if (= (type g_name) 'STR)
  (setq wbOK (if mgs T nil)))
(if wbOK
  (progn
; При отсутствии группы: Automation Error. Invalid argument XXX in Item
    (setq mg (vl-catch-all-apply 'vla-Item (list mgs g_name)))
    (if (/= (type mg) 'VLA-OBJECT)
      (progn
        (setq wbOK nil)
        (princ "\nНеверно имя группы адаптации (меню). ")
      );progn
    );if
  );progn
);if
; Имя группы правильное
; Проверяем, нет ли уже одноименной панели
(if wbOK
  (progn
    (if (= (type t_name) 'STR)
      (progn
        (setq tbs (vla-get-Toolbars mg))
; При отсутствии панели: Automation Error. Invalid argument XXX in Item
        (setq tb (vl-catch-all-apply 'vla-Item (list tbs t_name)))
        (if (not (vl-catch-all-error-p tb))
          (progn
            (setq wbOK nil); повтор имени
            (princ "\nПовтор имени панели. ")
          );progn
        );if
      );progn
      (progn
        (setq wbOK nil)
        (princ "\nНеверно имя панели. ")
      );progn
    );if
  );progn
);if
```

```
; Имя новой панели верно. Создаем новую панель
(if wbOK
  (progn
    (setq tb (vla-Add tbs t_name))
    (if tb
      (progn
    ; Добавление кнопок и установка пиктограмм
    ; Нужны проверки типов данных аргументов-списков
        (setq i -1)
        (repeat (length l_names)
          (setq i (1+ i))
          (setq tbitem (vla-AddToolbarButton tb
     i (nth i l_names) (nth i l_helps) (nth i l_macros))); кнопка i
          (vla-SetBitmaps tbitem (nth i l_icons) (nth i l_icons))
        );repeat
        ; Координаты панели на экране и количество рядов
        (vla-Float tb cx cy nrows)
        (princ (strcat "\nПанель " t_name
          " создана в группе " g_name ". "))
        (setq g_tb tb); сохранение в глобальной переменной
                      ; для дальнейшего использования
        (vlr-beep-reaction); звуковой сигнал
      );progn
      (princ (strcat "\nПанель с именем " t_name
        " уже существует в группе " g_name ". "))
    );if
  );progn
);if
; Освобождение памяти, занятой ненужными VLA-объектами
(mapcar (function (lambda (x)
  (if (and x (not (vlax-object-released-p x)))
    (vlax-release-object x)))) (list mgs mg tbs tbitem))
(princ)
);defun cr_tb
```

Функция `cr_tb` имеет следующие аргументы:

- `g_name` — имя группы адаптации (меню), в которую добавляется панель (должно быть строкой с именем одной из групп, загруженных в память). Например: `"ACAD"`;
- `t_name` — имя создаваемой панели (должно быть строкой и отличаться от всех имен панелей, уже существующих в группе `g_name`). Например: `"Dancing Queen"`;
- `l_names` — список из строк с названиями кнопок в новой панели (название выводится в желтом прямоугольнике при подведении курсора к кнопке). Например: `'("Dance1" "Dance2" "Dance3" "Dance4" "Dance5")`;

- `l_helps` — список из строк с подробными подсказками, выводимыми в строке состояния при подведении курсора к кнопке. Например: `'("Подсказка 1" "Подсказка 2" "Подсказка 3" "Подсказка 4" "Подсказка 5")`;
- `l_macros` — список из строк с макросами, которые выполняются при щелчке по кнопке. Например: `'("(alert\"New\") " "(alert\"Open\") " "(alert\"Save\") " "(alert\"SaveAs\") " "(alert\"Print\") ")`. Каждый макрос заканчивается пробелом (аналогом нажатия клавиши <Enter>);
- `l_icons` — список из строк с именами ресурсов пиктограмм, которые связываются с кнопками панели. Ресурсы должны быть иконками из DLL-файла ресурсов AutoCAD или BMP-файлами. Например: `'("RCDATA_16_NEW" "RCDATA_16_OPEN" "RCDATA_16_SAVE" "RCDATA_16_SAVE" "RCDATA_16_PRINT")` — имена ресурсов для кнопок `New`, `Open`, `Save` и `Print` из панели **Standard** (Стандартная). В версиях 2004 и 2005 такие же имена ресурсов. В версии 2002 вместо префикса `RCDATA_` используется префикс `ICON_`;
- `cx` — горизонтальная координата начального положения панели (неотрицательное целое число);
- `cy` — вертикальная координата начального положения панели (неотрицательное целое число);
- `nrows` — количество рядов в начальном положении (целое число от 1 до количества кнопок в панели).

Первое, что делает функция `cr_tb` — проверяет правильность имени группы, в которой необходимо создать панель. Имя группы передается через первый аргумент. Программа проверяет его тип на строковый, а затем проверяет, что группа с таким именем существует в текущем рисунке (она должна быть членом семейства `MenuGroups`). При проверке используется важная функция `vl-catch-all-apply`. В случае возникновения прерывания из-за ошибки, связанной с тем, что указанная группа не существует, эта функция возвращает не VLA-объект группы, а объект типа VL-CATCH-ALL-APPLY-ERROR. Подобная схема выявления ошибок уже использовалась нами в *приложении 4*.

Для проверки применяется такая конструкция:

```
(setq tb (vl-catch-all-apply 'vla-Item (list mgs g_name)))
```

После выполнения этого выражения можно спокойно проанализировать тип результата, не прерывая работу программы.

Аналогичная проверка осуществляется для аргумента `t_name`, который должен быть строкой. Но здесь функция `vl-catch-all-apply` применяется в обратном режиме: если ошибки нет, то это означает, что в семействе `Toolbars` уже есть панель с именем `t_name`, что для нас недопустимо!

После проверки правильности имени группы и имени панели инструментов программа переходит к созданию панели. Вначале с помощью метода `Add` создается VLA-объект пустой панели:

```
(setq tb (vla-Add tbs t_name))
```

Затем с помощью метода `AddToolbarButton` в панель добавляются кнопки, количество которых равно длине списка `l_names` (обращаем внимание, что для сокращения места в программе отсутствует проверка длин списков `l_helps`, `l_macros` и `l_icons` — если эти списки короче, то программа не сработает; кроме того, надо контролировать, что все элементы списков действительно являются строками).

Далее с помощью метода `SetBitmaps` к кнопкам привязываются изображения. В примере использованы пиктограммы кнопок панели инструментов **Standard** (Стандартная). В качестве имен малых и больших кнопок указаны одни и те же имена (это системой допускается). В результате на экране должна появиться панель (на рис. П5.1 показан пример панели **Dancing Queen** (Танцующая королева) с пятью кнопками).

Рис. П5.1. Панель **Dancing Queen**

VLA-объект построенной панели запоминаем в глобальной переменной `g_tb` для передачи в другие функции. Остальные VLA-объекты удаляются с помощью функции `vlax-release-object` и тем самым освобождается занятая ими память.

Следующая функция `move_tb` должна перемещать панель из одной точки экрана в другую в заданное количество шагов (слишком быстрое движение неинтересно). Текст функции приведен в листинге П5.2.

Листинг П5.2. Файл move_tb.lsp

```
; Приложение 5\move_tb.lsp
; © Н.Н.Полещук, 2006 г.
; В книге: Н.Н.Полещук, П.В.Лоскутов
;"AutoLISP и Visual LISP в среде AutoCAD"
; (издательство "БХВ-Петербург", 2006)
;
(defun move_tb (tb x1 y1 x2 y2 ntime / i x y)
(vla-Float tb x1 y1 1)
(setq i 0 x x1 y y1)
(repeat ntime
  (setq i (1+ i) x (+ x1 (/ (/ (* i (- x2 x1)) ntime))))
  (setq y (+ y1 (/ (/ (* i (- y2 y1)) ntime))))
  (vla-put-Left tb x)
  (vla-put-Top tb y)
);repeat
(vlr-beep-reaction)
);defun move_tb
```

Функция `move_tb` в цикле преобразует панель в один ряд кнопок и пересчитывает положение панели на экране. Аргумент `ntime` управляет временем, за которое панель пройдет путь между крайними точками. В конце маршрута с помощью функции `vlr-beep-reaction` выдается звуковой сигнал. Поскольку фоновое окно не успевает перерисовываться, то при движении панели на экране остается шлейф.

Следующая функция `dance_tb` (листинг П5.3) двигает панель на экране по маршруту, который задается списком из нескольких точек.

Листинг П5.3. Файл dance_tb.lsp

```
; Приложение 5\dance_tb.lsp
; © Н.Н.Полещук, 2006 г.
; В книге: Н.Н.Полещук, П.В.Лоскутов
;"AutoLISP и Visual LISP в среде AutoCAD"
; (издательство "БХВ-Петербург", 2006)
;
(defun dance_tb (tb plist ntime / wbOK p x0 y0 x1 y1)
(if (and (listp plist)(> (vl-list-length plist) 0))
  (progn
    (setq plist (vl-remove-if-not 'test plist))
    (setq x0 (vla-get-Left tb) y0 (vla-get-Top tb))
    (while plist
      (setq x1 (caar plist) y1 (cadar plist))
      (move_tb tb x0 y0 x1 y1 ntime)
      (setq x0 x1 y0 y1 plist (cdr plist))
    );while plist
  );progn
);if
);defun dance_tb
; Проверяем список на 2 элемента и оба числа (numberp)
; Вспомогательная функция
(defun test (p / wbOK)
  (setq wbOK (if (and (listp p) (= (vl-list-length p) 2)) T nil))
  (if wbOK
    (setq wbOK (and (numberp (car p)) (numberp (cadr p))))
  );if
);defun test    (T только для правильного списка)
```

В функции `dance_tb` сначала проверяется правильность списка `plist` и из него удаляются неправильные элементы (не представляющие двумерные точки). После этого многократно применяется функция перемещения `move_tb`.

Для получения полной картины осталось только привести программный код, который выполняет танец от начала и до конца (листинг П5.4).

Листинг П5.4. Файл show.lsp

```
; Приложение 5\show.lsp
; Параметры новой панели
(setq gn "ACAD" tn "Dancing Queen")
(setq ln (list "Dance1" "Dance2" "Dance3" "Dance4" "Dance5"))
(setq lh (list "Подсказка 1" "Подсказка 2" "Подсказка 3" "Подсказка 4"
  "Подсказка 5"))
(setq lm (list "(alert\"New\") " "(alert\"Open\") " "(alert\"Save\") "
  "(alert\"SaveAs\") " "(alert\"Print\") "))
(setq li (list "RCDATA_16_NEW" "RCDATA_16_OPEN" "RCDATA_16_SAVE"
  "RCDATA_16_SAVE" "RCDATA_16_PRINT"))
(setq k1 200 k2 150 kr 1)
(cr_tb gn tn ln lh lm li k1 k2 kr)
;;; Пример танца
(dance_tb g_tb (list '(100 100) '(500 200) '(0 400)) 10000)
;;; Удаление танцовщицы
(vla-Delete g_tb)
(vlax-release-object g_tb)
(setq g_tb nil)
(princ "\nПанель удалена. Концерт окончен. ")
(princ)
```

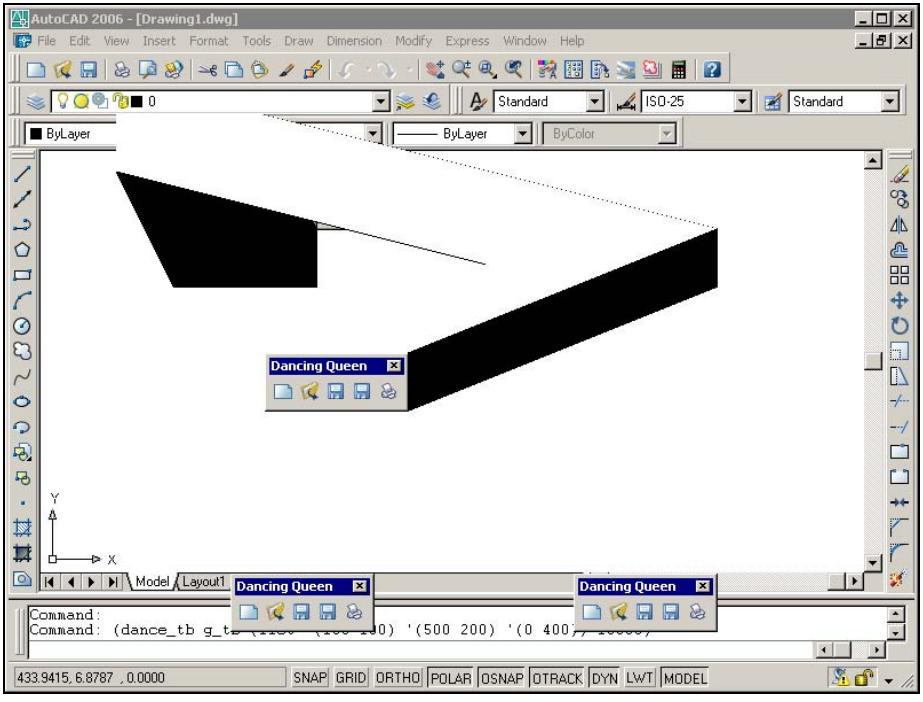


Рис. П5.2. Танцующая панель

На рис. П5.2 показан "танцевальный момент". Виден след от шлейфа панели. Любопытны изображения "мнимых" панелей, которые появляются в нижней части (объяснить их трудно).

Созданную панель можно и не удалять: она удалится при выходе из сеанса работы с системой AutoCAD. Панель существует только в оперативной памяти компьютера (почти что в нашем воображении...). Но свои задачи она исправно выполнит.

Панели, которые необходимы пользователю постоянно, рекомендуется не создавать динамически, как в рассмотренном нами примере, а загружать в версии 2006 через CUI-файл (в более ранних версиях — через MNU-файл).

ПРИЛОЖЕНИЕ 6

Пересечение объектов

При работе с геометрическими объектами рисунка очень часто приходится рассчитывать их точки пересечения (или хотя бы выяснять, не пересекаются ли примитивы). Решение этой трудоемкой задачи можно возложить на инструмент самой системы AutoCAD, а именно на метод `IntersectWith` технологии ActiveX.

В листинге П6.1 приведена функция `ent-int`, предназначенная для расчета точек пересечения двух примитивов или их продолжений (причем задавать продолжение можно только для одного объекта или сразу обоих).

Листинг П6.1. Файл ent-int.lsp

```
; Приложение 6\ent-int.lsp
; © Н.Н.Полещук, 2006 г.
; В книге: Н.Н.Полещук, П.В.Лоскутов
;"AutoLISP и Visual LISP в среде AutoCAD"
; (издательство "БХВ-Петербург", 2006)
;
; Листинг П6.1. Файл ent-int.lsp
;

(defun ent-int (e1 e2 m / plist pl1 pl2)
(vl-load-com)
(if (or (/= (type e1) 'ENAME) (/= (type e2) 'ENAME))
  (princ "\nНеверен тип аргументов. ")
  (progn
    ; Проверка 3-го аргумента
    (if (not (member m
      '(acExtendNone acExtendThisEntity
        acExtendOtherEntity acExtendBoth)))
      (setq m acExtendNone); по умолчанию и в случае ошибки -
    );if                   ; без продления
```

```
    ; Вычисление пересечения
    (setq plist (vla-IntersectWith
      (vlax-ename->vla-object e1) (vlax-ename->vla-object e2)
      m))
    ; Преобразование варианта в массив
    (setq plist (vlax-variant-value plist))
    ; Проверка правильности индексов массива
    (if (>= (vlax-safearray-get-u-bound plist 1)
            (vlax-safearray-get-l-bound plist 1))
    ; Преобразование в список точек
      (progn
        (setq pl1 (vlax-safearray->list plist)); плоский список
        (setq pl2 nil)
        (while (>= (vl-list-length pl1) 3)
          (setq pl2 (append pl2
            (list (list (car pl1) (cadr pl1) (caddr pl1)))))
          (setq pl1 (cdddr pl1))
        );while
      );progn
    );if
  );progn
);if
; Возвращаемое значение
pl2
);defun ent-int
```

Функция `ent-int` имеет следующие аргументы:

- `e1` — первый примитив;
- `e2` — второй примитив;
- `m` — признак продолжения примитивов.

Последний аргумент может принимать только следующие целочисленные значения, заданные в структуре типа перечисление с помощью зарезервированных символов:

- `acExtendNone` — примитивы перед пересечением не продлевать;
- `acExtendThisEntity` — продлить первый объект;
- `acExtendOtherEntity` — продлить второй объект;
- `acExtendBoth` — продлить оба примитива.

Если для какого-то из графических объектов задан признак продления, то крайние участки этого объекта перед вычислением виртуально максимально продлеваются. В таком случае какие-то из найденных точек пересечения могут оказаться лежащими не на самом примитиве, а на его продолжении.

Применяемый для расчета точек пересечения метод `IntersectWith` имеет свои особенности. Во-первых, возвращаемое значение является вариантом с типом безопасного массива. Во-вторых, при отсутствии пересечения не возвращается какой-нибудь простой признак (как `nil`), а устанавливается –1 в качестве верхнего индекса массива (при этом нижняя граница — 0). Поэтому для выявления случая непересечения приходится с помощью функций `vlax-safearray-get-u-bound` и `vlax-safearray-get-l-bound` вычислять верхний и нижний индексы и сравнивать их значения на непротиворечивость.

Поскольку в обычном случае возвращается вариант, то его приходится преобразовывать к привычному виду: вначале к безопасному массиву, затем к списку чисел (в нем нет деления на тройки координат точек), а в конце концов приводить к списку точек.

В листинге П6.2 показан пример использования функции `ent-int`.

Листинг П6.2. Файл ex6.lsp

```
; Приложение 6\ex6.lsp
; В книге: Н.Н.Полещук, П.В.Лоскутов
;"AutoLISP и Visual LISP в среде AutoCAD"
; (издательство "БХВ-Петербург", 2006)

; Интерактивный выбор двух объектов
(setq ent1 (car (entsel)) ent2 (car (entsel)))
(setq pts (ent-int ent1 ent2 acExtendNone))
(if pts
  (progn
    (princ (strcat "\nТочки пересечения (" (itoa (length pts)) "): "))
    (mapcar 'princ pts)
  );progn
  (princ "\nОбъекты не пересекаются. ")
);if
```

Пример результата выполнения файла ex6.lsp:

```
Точки пересечения (12): (232.615 200.911 0.0)(269.569 139.946 0.0)(120.613
144.91 0.0)(154.743 199.435 0.0)(132.002 176.799 0.0)(178.601 209.332
0.0)(140.416 85.1491 0.0)(123.915 112.851 0.0)(185.721 62.159 0.0)(267.478
118.451 0.0)(246.331 82.105 0.0)(221.641 66.545 0.0)
```

ПРИЛОЖЕНИЕ 7

Очистка расширенных данных

Когда вы храните вместе с примитивом расширенные данные, то может возникнуть вопрос замены данных, связываемых с каким-то приложением *(о расширенных данных см. разд. 1.14)*.

Естественная попытка заменить соответствующий подсписок в полном списке данных примитива с последующим обновлением объекта заканчивается тем, что расширенные данные накапливаются, а не заменяются. Поэтому перед тем как заменить, необходимо данные приложения удалить (не трогая, конечно, данных других приложений).

В листинге П7.1 приведена функция `clean_xdata`, которая удаляет из примитива расширенные данные указанного приложения.

Листинг П7.1. Файл clean_xdata.lsp

```
; Приложение 7\clean_xdata.lsp
; © Н.Н.Полещук, 2006 г.
; В книге: Н.Н.Полещук, П.В.Лоскутов
;"AutoLISP и Visual LISP в среде AutoCAD"
; (издательство "БХВ-Петербург", 2006)
;
(defun clean_xdata (ent app / le le1)
(if (and (= (type ent) 'ENAME) (= (type app) 'STR))
  (progn
; Читаем список данных с расширенными данными приложения app
    (setq le (entget ent (list "*")))
; Удаляем расширенные данные приложения app,
; заменяя их на частично пустой список
    (setq le1 (subst (cons -3 (list (list app))) (assoc -3 le) le))
; Обновляем примитив
    (entmod le1)
  );progn
```

```
  (princ "\nНеверен тип аргументов. ")
);
(princ)
);clean_xdata
```

Функция clean_xdata имеет два аргумента:

- ent — примитив;
- app — строка с именем приложения, расширенные данные которого необходимо удалить.

Сначала с помощью функции entget получается полный список данных примитива (с расширенными данными всех приложений):

```
(setq le (entget ent (list "*")))
```

Сами расширенные данные сосредоточены в подсписке с DXF-кодом –3, например:

```
(-3 ("MyApp" (1000 . "Yt") (1040 . 45.8))
    ("MyApp1" (1070 . 123) (1000 . "Ztt1")))
```

Если составить такой же подсписок с DXF-кодом –3, но в нем расширенные данные приложения будут состоять только из имени приложения (например, (-3 ("MyApp1"))), то при обновлении функцией entmod с помощью списка, в котором к основным данным примитива добавлен приведенный подсписок, из примитива будут удалены расширенные данные приложения MyApp1.

В листинге П7.2 показан пример программы создания объекта с расширенными данными двух приложений, которые затем поэтапно очищаются с помощью функции clean_xdata.

Листинг П7.2. Файл test.lsp

```
; Приложение 7\test.lsp
; В книге: Н.Н.Полещук, П.В.Лоскутов
;"AutoLISP и Visual LISP в среде AutoCAD"
; (издательство "БХВ-Петербург", 2006)
;
; Регистрируем приложения MyApp и MyApp1
(regapp "MyApp")
(regapp "MyApp1")

; Создаем примитив CIRCLE с расширенными данными обоих приложений
(setq e0 (entmakex  (list '(0 . "CIRCLE") '(10 200.0 50.0 0.0) '(40 . 100.0)
  '(-3 ("MyApp" (1000 . "Yt")(1040 . 45.8))
       ("MyApp1" (1070 . 123)(1000 . "Ztt1"))))))
; Список, возвращаемый выражением (entget e0 '("*"))
;((-1 . <Entity name: 7ef65e48>) (0 . "CIRCLE")
;(330 . <Entity name: 7ef65d18>) (5 . "49") (100 . "AcDbEntity")
```

```
;(67 . 0) (410 . "Model") (8 . "0") (100 . "AcDbCircle")
;(10 200.0 50.0 0.0) (40 . 100.0) (210 0.0 0.0 1.0)
;(-3 ("MyApp" (1000 . "Yt") (1040 . 45.8))
;("MyApp1" (1070 . 123) (1000 . "Ztt1"))))

; Очищаем данные приложения MyApp1
(clean_xdata e0 "MyApp1")
; Список данных примитива
;((-1 . <Entity name: 7ef65e48>) (0 . "CIRCLE")
;(330 . <Entity name: 7ef65d18>)
;(5 . "49") (100 . "AcDbEntity") (67 . 0)
;(410 . "Model") (8 . "0") (100 . "AcDbCircle")
;(10 200.0 50.0 0.0) (40 . 100.0) (210 0.0 0.0 1.0)
;(-3 ("MyApp" (1000 . "Yt") (1040 . 45.8))))

; Очищаем данные приложения MyApp
(clean_xdata e0 "MyApp")
; Список данных примитива
; ((-1 . <Entity name: 7ef65e48>) (0 . "CIRCLE")
;(330 . <Entity name: 7ef65d18>) (5 . "49") (100 . "AcDbEntity")
;(67 . 0) (410 . "Model") (8 . "0") (100 . "AcDbCircle")
;(10 200.0 50.0 0.0) (40 . 100.0) (210 0.0 0.0 1.0))
```

Программа, приведенная в листинге П7.2, создает окружность с расширенными данными приложений `MyApp` и `MyApp1`. Далее из примитива удаляются сначала данные приложения `MyApp1` и приводится список оставшихся данных, а затем удаляются расширенные данные приложения `MyApp`. Результирующий список показывает, что очистка выполнена верно.

ПРИЛОЖЕНИЕ 8

Метод *CopyObjects*

Метод[1] `CopyObjects`, в отличие от метода `Copy`, позволяет копировать (клонировать) как графические, так и неграфические объекты (словари, таблицы и т. д.). При использовании метода для копирования объектов в одном рисунке целесообразно применять его для создания блоков из существующих объектов, копирования объектов с переносом из одного пространства в другое или переопределения существующих блоков (добавления объектов в существующий блок).

Но наиболее интересным является применение метода для копирования неграфических объектов в другие документы. В этом случае возвращаемым значением функции `vla-CopyObjects` является пустой безопасный массив (т. к. графические примитивы не участвуют) и для определения результатов копирования необходимо исследование специального объекта, содержащего пары `IDPair`, которые создаются во время применения метода.

Благодаря связи между исходными и скопированными объектами, которая сохраняется в специальных объектах `IDPair`, пользователь может анализировать успешность копирования объекта и получать ассоциативные списки ID объектов исходного и целевого документа.

Работу метода `CopyObjects` мы будем исследовать на примере функции `modify_identical_objects`, которая позволяет копировать из активного (исходного) документа словари, стили текста, размерные стили, слои, блоки во все открытые (целевые) документы, изменяя значения свойств этих объектов при совпадении имен. Функция окажет помощь в случаях, если документы были ошибочно созданы без использования шаблона или обнаружена ошибка в использованном шаблоне.

Рассматриваемая функция копирования базируется на четырех приведенных в листингах П8.1—П8.4 вспомогательных функциях.

---

[1] Материал приложения подготовил Э. В. Смолянка.

Представленная в листинге П8.1 функция `copy_non_graphic` получает в качестве аргумента указатель на активный документ, сканирует базу этого документа, определяет объекты, которые будут копироваться в целевые рисунки (слои, блоки, типы линий, текстовые стили, размерные стили) и выполняет копирование этих объектов в целевые документы. При определении объектов, копирование которых будет выполняться, функция удаляет из набора те, имена которых содержат символ "*" (блоки `*ModelSpace`, `*PaperSpace`, анонимные блоки) и символ "|" (слои и блоки внешних ссылок, вставленных в исходный рисунок).

Листинг П8.1. Функция `copy_non_graphic`

```
; Приложение 8\copy_non_graphic.lsp
; © Э.В.Смолянка, 2006 г.
; В книге: Н.Н.Полещук, П.В.Лоскутов
;"AutoLISP и Visual LISP в среде AutoCAD"
; (издательство "БХВ-Петербург", 2006)
;
(defun copy_non_graphic
       (adoc / function_name idpair item2 list_idpair list_objects)
  (setq function_name '(vla-get-Layers vla-get-Linetypes
                        vla-get-Blocks vla-get-TextStyles
                        vla-get-DimStyles)
    list_objects (mapcar (function (lambda (x / lst)
              (vlax-for item (eval (list x adoc))
                (if(not
                  (and (vlax-property-available-p item "Name")
                    (wcmatch (vla-get-Name item) "*|*,'**")))
                    (setq lst (cons item lst))))
                      (if lst
                        (vlax-safearray-fill
                          (vlax-make-safearray
                            vlax-vbObject
                             (cons 0 (1- (length lst))))
                            lst))))
                       function_name))
  (vlax-for item2 (vla-get-Documents (vlax-get-acad-object))
    (if (equal (vla-get-Active item2) :vlax-false)
      (setq list_idpair
        (cons (apply (function append)
          (mapcar (function (lambda (x y)
            (if y
              (progn (vla-CopyObjects adoc y
                (eval (list x item2))
                             'idpair)
```

```
                        (vlax-safearray->list idpair)))))
                          function_name
                          list_objects))
                      list_idpair))))
  (apply (function append) list_idpair))
```

Возвращаемым значением функции является список тех пар (`IDPair`), которые соответствуют всем копируемым объектам, и дополнительных пар, возникших при использовании метода `CopyObjects` (классы объектов, требующих дополнительных пар, приведены в табл. П8.1).

Результат выполнения функции `copy_non_graphic` (значение переменной `idpair`) поступает в качестве аргумента в представленную в листинге П8.2 функцию `idpair_object`, которая проводит анализ `IDPair`, удаляет из списка дополнительные `IDPair` и пары, копирование которых прошло успешно (значение свойства `IsCloned` этих пар равно `:vlax-true`). Из `IDPair` методом `ObjectIDToObject` определяются указатель на объект исходного рисунка, идентификатор которого является значением свойства `Key` пары, и указатель на объект целевого рисунка, идентификатор которого является значением свойства `Value` пары.

Возвращаемым значением функции `idpair_object` является список, элементами которого являются списки вида:

```
(<исходный объект> (<целевой объект> <IsCloned> <IsOwnerXlated>))
```

Эти списки содержат объекты, копирование которых не произошло из-за совпадения имен объектов в исходном и целевом рисунках.

Листинг П8.2. Функция `idpair_object`

```
; Приложение 8\idpair_object.lsp
; © Э.В.Смолянка, 2006 г.
; В книге: Н.Н.Полещук, П.В.Лоскутов
;"AutoLISP и Visual LISP в среде AutoCAD"
; (издательство "БХВ-Петербург", 2006)
;
(defun idpair_object (adoc idpar)
  (vl-remove-if
    (function (lambda (s)
              (or (wcmatch (vla-get-ObjectName (car s))
                  "AcDbDictionary*,*Table,*Holder")
                  (equal (cadadr s) :vlax-true))))
    (vl-remove nil
      (mapcar (function (lambda (y)
         (list (vla-ObjectIDToObject adoc (vla-get-Key y))
            (list (vla-ObjectIDToObject adoc (vla-get-Value y))
```

```
                (vla-get-IsCloned y)
                (vla-get-IsOwnerXlated y)))))
        idpar)))
)
```

Функция list_property_objects (листинг П8.3), используя функцию pl:get-protect-names (листинг П8.4) и принимая в качестве аргумента указатель на объект исходного рисунка, определяет все возможные изменяемые свойства этого объекта, считывает их значения и сохраняет их в списке для дальнейшего использования главной функцией modify_identical_objects. При считывании значений свойств объекта нужно быть осторожным, т. к. некоторые свойства объект может иметь только при определенных условиях. Например, если объект Block не является внешней ссылкой, то попытка считать его свойство Path вызывает ошибку (not applicable), хотя проверка на наличие этого свойства функцией vlax-property-available-p возвращает :vlax-true. Для предотвращения такой ошибки в функции list_property_objects применена функция vl-catch-all-apply с последующим удалением значений свойств из возвращаемого функцией результата.

Листинг П8.3. Функция list_property_object

```
; Приложение 8\list_property_object.lsp
; © Э.В.Смолянка, 2006 г.
; В книге: Н.Н.Полещук, П.В.Лоскутов
;"AutoLISP и Visual LISP в среде AutoCAD"
; (издательство "БХВ-Петербург", 2006)
;
(defun list_property_object (vla_object propnames)
  (vl-remove  nil
    (mapcar (function (lambda(x / res)
        (if
          (and (vlax-property-available-p vla_object x T)
            (not (vl-catch-all-error-p
              (setq res (vl-catch-all-apply
                (read (strcat "vla-get-" x))
                       (list vla_object))))))
           (list (read (strcat "vla-put-" x)) res))))
      propnames))
)
```

Листинг П8.4. Функция pl:get-protect-names

```
; Приложение 8\pl_get-protect-names.lsp
; © П.В.Лоскутов, 2006 г.
```

```
; В книге: Н.Н.Полещук, П.В.Лоскутов
;"AutoLISP и Visual LISP в среде AutoCAD"
; (издательство "БХВ-Петербург", 2006)
;
(defun pl:get-protect-names (filter rev)
  (setq filter (strcase filter))
  (apply (if rev
           (function vl-remove-if)
           (function vl-remove-if-not))
         (list (function (lambda (x) (wcmatch x filter)))
             (atoms-family 1))))
```

Представленная в листинге П8.5 функция `modify_identical_objects`, используя все вышеперечисленные функции, изменяет значения свойств объектов всех целевых документов в соответствии с исходным при совпадении имен этих объектов (т. е. при неуспешном копировании).

Листинг П8.5. Функция `modify_identical_objects`

```
; Приложение 8\modify_identical_objects.lsp
; © Э.В.Смолянка, 2006 г.
; В книге: Н.Н.Полещук, П.В.Лоскутов
;"AutoLISP и Visual LISP в среде AutoCAD"
; (издательство "БХВ-Петербург", 2006)
;
(defun modify_identical_objects ( / adoc prop)
(setq adoc (vla-get-ActiveDocument (vlax-get-acad-object))
  prop (mapcar (function (lambda (y) (substr y 9)))
  (vl-remove-if-not
    (function (lambda (x) (= (type (eval (read x))) 'SUBR)))
    (pl:get-protect-names "vla-put-*" nil))))
  (foreach i (idpair_object adoc (copy_non_graphic adoc))
    (foreach z (list_property_object (car i) prop)
      (vl-catch-all-apply (car z) (list (caadr i) (cadr z)))))
  (princ))
```

В табл. П8.1 перечислены типы классов объектов, для которых при клонировании создаются дополнительные пары.

Важная дополнительная информация.

- При копировании неграфических объектов из одного рисунка в другой функция `vla-CopyObjects` производит проверку на совпадение имен объектов. В случае если имена совпадают, копирование не происходит, но ID-пара (`IDPair`) создается.

- В случае копирования объекта `Insert` из одного рисунка в другой необходимо проверять таблицу блоков рисунка, в который происходит копирование (целевого рисунка) на наличие подобного блока, т. к. вставленный объект `Insert` при копировании переопределится в соответствии с графическим представлением блока целевого рисунка.
- При копировании графического объекта в другой рисунок автоматически создается слой с такими же свойствами, как и в исходном рисунке, загружается тип линий, которым нарисован объект.
- Если слой с таким именем в целевом рисунке уже существует и копируемый объект имеет значения цвета, типа линий, веса ByLayer (ПоСлою), то этот объект наследует свойства существующего в целевом рисунке слоя.

Таблица П8.1. *Классы объектов, для которых создаются дополнительные* `IDPair`

Тип объекта	Тип объекта
`AcDbBlockTable`	`AcDbLinetypeTableRecord`
`AcDbBlockTableRecord`	`AcDbPlaceHolder`
`AcDbDictionary`	`AcDbRegAppTable`
`AcDbDictionaryWithDefault`	`AcDbTextStyleTable`
`AcDbDimStyleTable`	`AcDbTextStyleTableRecord`
`AcDbFontTable`	`AcDbUCSTable`
`AcDbLayerTable`	`AcDbViewportTable`
`AcDbLayerTableRecord`	`AcDbViewTable`
`AcDbLinetypeTable`	`AcDbVXTable`

ПРИЛОЖЕНИЕ 9

Описание компакт-диска

Большая часть примеров, рассмотренных в книге, размещена на сопроводительном компакт-диске. Местоположение программных файлов и их имена приведены прямо в тексте листингов.

На диске находятся также дополнительные примеры, подготовленные Э. В. Смолянкой.

Перечень материалов

Материалы диска распределены по папкам глав, а внутри — по папкам проектов, которые указаны в листингах. В каждой папке имеется файл readme.txt, содержащий краткие пояснения.

В табл. П9.1 приведен перечень файлов, которые можно найти на диске.

Таблица П9.1. *Перечень файлов, размещенных на сопроводительном компакт-диске*

№	Папка главы	Папка проекта	Файлы
1	Глава 01	—	Листинг 1.01.lsp, Листинг 1.02.lsp, Листинг 1.03.lsp, Листинг 1.04.lsp, Листинг 1.05.lsp, Листинг 1.06.lsp, Листинг 1.07.lsp, Листинг 1.08.lsp, Листинг 1.09.lsp, Листинг 1.10.lsp, Листинг 1.11.lsp, Листинг 1.12.lsp, Листинг 1.13.lsp, Листинг 1.14.lsp, Листинг 1.15.lsp, Листинг 1.16.lsp, Листинг 1.17.lsp, Листинг 1.18.lsp, Листинг 1.19.lsp, Листинг 1.20.lsp, Листинг 1.21.lsp, Листинг 1.22.lsp, Листинг 1.23.lsp, Листинг 1.24.lsp, Листинг 1.25.lsp, Листинг 1.26.lsp, Листинг 1.27.lsp, Листинг 1.28.lsp, Листинг 1.29.lsp, Листинг 1.30.lsp, Листинг 1.31.lsp

Таблица П9.1 *(продолжение)*

№	Папка главы	Папка проекта	Файлы
2	Глава 03	Book01	d_tpos.dcl, show_tpos.lsp
3	Глава 03	Book02	sample1.lsp, sample2.lsp, sample3.lsp, sample4.lsp, sample5.lsp, sample6.lsp, sample7.lsp, sample8.lsp, sample9.lsp, sample10.lsp, sample11.lsp, sample12.lsp, sample13.lsp
4	Глава 03	Book03	polz01.txt, polz02.txt, user.dcl, usn.lsp
5	Глава 03	Book04	long_popuplist.dcl, long_popuplist.lsp
6	Глава 03	Book05	all.txt, My.slb, Name1.sld, Name2.sld, Name3.sld, Name4.sld, Name5.sld, Name6.sld, Name7.sld, slide.dcl, slide.lsp
7	Глава 03	Book06	imcolor.dcl, imcolor.lsp
8	Глава 03	Book07	imvector.dcl, imvector.lsp
9	Глава 03	Book08	tex.dcl, tex.lsp
10	Глава 03	Book09	twodia.dcl, twodia.lsp
11	Глава 03	Book10	pass.dcl
12	Глава 06	Book01	begin_activex.lsp, sample.lsp
13	Глава 06	Book02	ldata.lsp, lfilters.lsp
14	Глава 06	Book03	anydoc_clayer.lsp, item_schmenu.lsp, menu_names.lsp
15	Глава 09	Book01	reac1.lsp
16	Глава 09	Book02	cback2.lsp, reac2.lsp
17	Глава 09	Book03	cback3.lsp, reac3.lsp
18	Глава 09	Book04	cback4.lsp, reac4.lsp
19	Глава 09	Book05	cback5.lsp, reac5.lsp
20	Глава 09	Book06	cback6.lsp, reac6.lsp
21	Глава 10	Book01	PL_Mlist_Creat.lsp, PL_Mlist_Func.lsp, PL_Mlist_Set.lsp, PL_MTI_Func.lsp, PL_MTX_Creat.lsp, PL_MTX_Ev.lsp, PL_MTX_Func.lsp, PL_MTX_Func2.lsp, PL_MTX_IDX.lsp, PL_MTX_Set.lsp, PL_ReadFile.lsp, PL_String-_List.lsp, PL_String-Rep.lsp
22	Глава 10	Book02	pl_obj-filter-select-all.lsp, pl_obj-filter-select-manual.lsp
23	Глава 10	Book03	pl_demo.lsp

Таблица П9.1 *(окончание)*

№	Папка главы	Папка проекта	Файлы
24	Глава 10	Book04	ex11_break_connect.lsp, ex11_set_connect.lsp, readex11.xls, readex11_com.lsp
25	Приложение 3	—	pbar.lsp, progress.dcl
26	Приложение 4	—	checknum.lsp, lights.dcl
27	Приложение 5	—	cr_tb.lsp, dance_tb.lsp, move_tb.lsp, show.lsp
28	Приложение 6	—	ent-int.lsp, ex6.lsp
29	Приложение 7	—	clean_xdata.lsp, test.lsp
30	Приложение 8	—	copy_non_graphic.lsp, idpair_object.lsp, list_property_object.lsp, modify_identical_objects.lsp, pl_get-protected-names.lsp

Использование материалов

При использовании в своих разработках материалов с компакт-диска не забудьте сослаться на их авторов.

Литература

1. Зуев С. А., Полещук Н. Н. САПР на базе AutoCAD — как это делается. — СПб.: БХВ-Петербург, 2004. — 1168 с. (+CD-ROM)
2. Полещук Н. Н. AutoCAD: разработка приложений, настройка и адаптация. — СПб.: БХВ-Петербург, 2006. — 992 с. (+CD-ROM)
3. Полещук Н. Н. AutoCAD 2002. — СПб.: БХВ-Петербург, 2003. — 1200 с.
4. Полещук Н. Н. AutoCAD 2004. — СПб.: БХВ-Петербург, 2004. — 976 с.
5. Полещук Н. Н. AutoCAD 2004: разработка приложений и адаптация. — СПб.: БХВ-Петербург, 2004. — 624 с.
6. Полещук Н. Н. AutoCAD 2005. — СПб.: БХВ-Петербург, 2005. — 944 с.
7. Полещук Н. Н. Visual LISP и секреты адаптации AutoCAD. — СПб.: БХВ-Петербург, 2001. — 576 с.
8. Poleshchuk N. AutoCAD Developer's Guide to Visual LISP. — Wayne: A-LIST, LLC, 2001. — 582 с.
9. Stein D. The Visual LISP Developers Bible — 2003 Edition. — Электронное издание, 2003. — 183 с.
10. Świszczowski S. AutoLISP. — Warszawa: MIKOM, 2001. — 272 с.

Предметный указатель

A

acExtendBoth 879
acExtendNone 879
acExtendOtherEntity 879
acExtendThisEntity 879
AHP-файл 314
ARX-приложение 171

B

Blackboard 185, 230
Bulge 855

C

Callback function 759
CHM-файл 313
CNT-файл 311
COM 815
CUI-файл 877

D

DCL-файл 232, 234
DVB-приложение 177
DXF-код 44, 66, 116, 136, 844, 845
DXF-формат 844
◊ секция 850

E

Excel 815

F

FAS-приложение 223, 227
FAS-файл 214

H

HELP-файл 309
HTML-файл 313
HWND 779

L

LISP-выражение 6

M

MNL-файл 306
MNU-файл 877

O

OLE_COLOR 553, 607, 608, 634, 655, 724

P

PRJ-файл 225
Progress bar 861
PRV-файл 228

R

RGB 285

S

Safearray 331

V

Vlax-vbInteger 331
Vlax-vbVariant 331
VLA-объект 326
VLR-объект 753
VLX-приложение 223, 228

W

Wild-card-characters 60
Winhlp32 313

X

X-запись 118, 149, 152, 342, 347

А

Адаптация 294
◊ группа 295
Анимация программы 225
Аргумент 8, 155, 180
Атом 31, 70
Атрибут 236, 239
◊ action 251, 263, 291
◊ alignment 251, 258
◊ big_increment 256
◊ children_alignment 240, 258
◊ children_fixed_height 241, 258
◊ children_fixed_width 241, 258
◊ color 253
◊ edit_limit 252
◊ edit_width 252
◊ fixed_height 258
◊ fixed_width 251, 252, 253, 258
◊ fixed_width_font 253
◊ height 240, 253
◊ initial_focus 240
◊ key 240
◊ label 240, 256, 258
◊ layout 256
◊ list 274
◊ max_value 256
◊ min_value 256
◊ multiple_select 253
◊ password_char 293
◊ small_increment 256
◊ value 240, 255, 257, 265
◊ width 240

Б

Баланс скобок 7
Безопасный массив 331
Библиотека:
◊ слайдов 282
◊ типов 340
Блок 859
◊ анонимный 860
◊ динамический 860
Блокнот 185

В

Вариант 328
Вектор выдавливания 53
Вершина 114
Вид 136
Владелец 761, 786
Внедокументное пространство имен 230
Возвращаемое значение 8
Вывод:
◊ в бокс экранного меню 187
◊ в строку состояния 187
Выражение 6
Высота 847

Г

Группа 122

Д

Данные расширенные 139
Диалоговое окно:
◊ вложенное 291
◊ временный выход 289
Диалоговые окна 232
Дигитайзер 195
Директива 235
◊ описания диалога 238
◊ описания элемента 245
Документ неактивный 356

З

Загрузка:
◊ LISP-программы 223
◊ LISP-файла во все рисунки 14
◊ приложения 132, 233
Знаки специальные групповые 60

И

Именное пространство 173, 174
◊ внедокументное 185
◊ документа 181
Имитация нажатия <Enter> 16
Имя примитива 53
Индикатор 861
Инструкция @include 238
Интерпретатор 5

К

Клавиша:
◊ <F1> 315
◊ <Shift> 674
Класс:
◊ AcDbDictionary 150
◊ AcDbSymbolTable 134
Ключевое слово dialog 238
Кнопка 251
◊ макрос 870
◊ настройка изображения 870
◊ подсказка 870
Количество наборов 122
Команда:
◊ 3D 175
◊ ACADINFO 826
◊ ARRAY 326
◊ ATTDEF 195
◊ ATTEXT 195
◊ AUDIT 847
◊ BHATCH 195
◊ BLOCK 195
◊ BROWSER 573
◊ COLOR 16, 195
◊ COPY 326
◊ DBLCLKEDIT 793
◊ DIMREGEN 197
◊ DXBIN 549
◊ DXFIN 549
◊ ERASE 792
◊ EXPRESSTOOLS 175
◊ GRAPHSCR 187
◊ GROUP 122
◊ IMAGE 195
◊ IMAGEADJUST 195
◊ INSERT 195, 792, 848, 859
◊ LAYER 195
◊ LAYERP 197
◊ LAYERPMODE 197
◊ LINE 15
◊ LINETYPE 195
◊ LIST 117
◊ MIRROR 326
◊ MSLIDE 282
◊ MTEXT 195, 717
◊ MVSETUP 175
◊ OPEN 184, 788
◊ PEDIT 126, 699
◊ PFACE 857, 858
◊ PLOT 195
◊ PSOUT 681
◊ QNEW 689

◊ REDRAW 186, 190
◊ REGEN 187, 190
◊ RENAME 195
◊ SAVEAS 845, 849
◊ STRETCH 140
◊ STYLE 195
◊ TASKBAR 92
◊ TEXTSCR 187
◊ TOOLBAR 195
◊ UNDO 776, 793
◊ VBASTMT 184
◊ VIEW 195
◊ VLIDE 202
◊ VLISP 202
◊ WBLOCK 777
◊ XREF 848
◊ ZOOM 16, 522, 523, 524
◊ создание 158
Командная строка 8
Комментарии 9
Компиляция 161, 162
Константа:
◊ списковая 11
◊ строковая 11
Контроль ввода 864
Координаты видовых экранов 44
Кривая 351

М

Макрос 358
◊ acedRegCmds 336
Массив 331
Меню 294
◊ Visual LISP 202
◊ группа 295
◊ заголовок 296
◊ контекстное 304, 357
◊ макрос пункта 296
◊ раздел 295
◊ разделитель пунктов 296
◊ структура пункта 296
Метка 236, 846, 853
Метод:
◊ Activate 362
◊ Add 362, 873
◊ Add3DFace 365
◊ Add3DMesh 365
◊ Add3DPoly 366
◊ AddArc 366
◊ AddAttribute 367
◊ AddBox 367
◊ AddCircle 368
◊ AddCone 368
◊ AddCustomInfo 368
◊ AddCustomObject 369
◊ AddCylinder 369
◊ AddDim3PointAngular 369
◊ AddDimAligned 370
◊ AddDimArc 371
◊ AddDimDiametric 371
◊ AddDimOrdinate 371
◊ AddDimRadial 372
◊ AddDimRadialLarge 372
◊ AddDimRotated 373
◊ AddEllipse 373
◊ AddEllipticalCone 374
◊ AddEllipticalCylinder 374
◊ AddExtrudedSolid 374
◊ AddExtrudedSolidAlongPath 375
◊ AddFitPoint 375
◊ AddHatch 376
◊ AddItems 377
◊ AddLeader 377
◊ AddLightWeightPolyline 378
◊ AddLine 378
◊ AddMenuItem 358, 379
◊ AddMInsertBlock 379
◊ AddMLine 380
◊ AddMText 380
◊ AddObject 381
◊ AddPoint 381
◊ AddPolyfaceMesh 382
◊ AddPolyline 382
◊ AddPViewport 383
◊ AddRaster 383
◊ AddRay 384
◊ AddRegion 384
◊ AddRevolvedSolid 385
◊ AddSeparator 385

Метод *(прод.)*:
◊ AddShape 386
◊ AddSolid 386
◊ AddSphere 386
◊ AddSpline 387
◊ AddSubMenu 387
◊ AddTable 388, 824
◊ AddText 388
◊ AddTolerance 389
◊ AddToolbarButton 389, 874
◊ AddTorus 390
◊ AddTrace 390
◊ AddVertex 391
◊ AddWedge 391
◊ AddXline 391
◊ AddXRecord 392
◊ AngleFromXAxis 392
◊ AngleToReal 392
◊ AngleToString 393
◊ AppendInnerLoop 393
◊ AppendItems 394
◊ AppendOuterLoop 394
◊ AppendVertex 395
◊ ArrayPolar 326, 395
◊ ArrayRectangular 326, 396
◊ AttachExternalReference 396
◊ AttachToolbarToFlyout 397
◊ AuditInfo 398
◊ Bind 398
◊ Block 398
◊ Boolean 399
◊ CheckInterference 399
◊ Clear 399
◊ ClearSubSelection 400
◊ ClearTableStyleOverrides 400
◊ ClipBoundary 400
◊ Close 401
◊ Connect 402
◊ ConvertToAnonymousBlock 402
◊ ConvertToStaticBlock 403
◊ Copy 326, 403
◊ CopyFrom 403
◊ CopyObjects 404, 621, 624, 627, 884
◊ CopyProfile 404
◊ CreateEntry 405
◊ CreateLabel 405
◊ CreateLink 406
◊ CreateTypedArray 407
◊ Delete 358, 408
◊ DeleteCellContent 408
◊ DeleteColumns 408
◊ DeleteConfiguration 409
◊ DeleteFitPoint 409
◊ DeleteProfile 409
◊ DeleteRows 410
◊ Detach 410
◊ Disconnect 410
◊ Display 411
◊ DisplayPlotPreview 411
◊ DistanceToReal 411
◊ Dock 412
◊ ElevateOrder 412
◊ EndUndoMark 413
◊ Erase 413
◊ Eval 184, 413
◊ Evaluate 414
◊ Evaluate (Excel) 823
◊ Explode 414
◊ Export 414
◊ ExportProfile 415
◊ FieldCode 416
◊ Float 416
◊ GenerateLayout 416
◊ GenerateUsageData 417
◊ GetAcadState 417
◊ GetAlignment 417
◊ GetAllProfileNames 418
◊ GetAngle 418
◊ GetAttachmentPoint 418
◊ GetAttributes 419
◊ GetAutoScale 419
◊ GetBackgroundColor 420
◊ GetBackgroundColorNone 420
◊ GetBitmaps 420
◊ GetBlockAttributeValue 421
◊ GetBlockRotation 421
◊ GetBlockScale 421
◊ GetBlockTableRecordId 422
◊ GetBoundingBox 422

◊ GetBulge 422
◊ GetCanonicalMediaNames 423
◊ GetCellAlignment 423
◊ GetCellBackgroundColor 423
◊ GetCellBackgroundColorNone 424
◊ GetCellContentColor 424
◊ GetCellExtents 424
◊ GetCellGridColor 425
◊ GetCellGridLineWeight 426
◊ GetCellGridVisibility 426
◊ GetCellStyleOverrides 426
◊ GetCellTextHeight 427
◊ GetCellTextStyle 427
◊ GetCellType 427
◊ GetColor 428
◊ GetColumnWidth 428
◊ GetConstantAttributes 428
◊ GetContentColor 429
◊ GetControlPoint 429
◊ GetCorner 430
◊ GetCustomByIndex 430
◊ GetCustomByKey 430
◊ GetCustomScale 431
◊ GetDistance 431
◊ GetDynamicBlockProperties 432
◊ GetEntity 432
◊ GetExtensionDictionary 145, 344, 432
◊ GetFieldId 433
◊ GetFitPoint 433
◊ GetFont 434
◊ GetFullDrawOrder 435
◊ GetGridColor 435
◊ GetGridLineWeight 436
◊ GetGridSpacing 436
◊ GetGridVisibility 437
◊ GetInput 437
◊ GetInteger 437
◊ GetInterfaceObject 438
◊ GetInvisibleEdge 438
◊ GetKeyword 438
◊ GetLabelTemplates 439
◊ GetLinks 439
◊ GetLinksByKey 440
◊ GetLinksByLabelTemplate 441
◊ GetLinksBySQLCond 441
◊ GetLinkTemplates 442
◊ GetLocaleMediaName 442
◊ GetLoopAt 443
◊ GetMinimumColumnWidth 443
◊ GetMinimumRowHeight 443
◊ GetName 444
◊ GetObject 444
◊ GetOrientation 444
◊ GetPaperMargins 445
◊ GetPaperSize 445
◊ GetPlotDeviceNames 446
◊ GetPlotStyleTableNames 446
◊ GetPoint 446
◊ GetProjectFilePath 447
◊ GetQueries 447
◊ GetReal 448
◊ GetRelativeDrawOrder 448
◊ GetRemoteFile 449
◊ GetRowHeight 449
◊ GetRowType 449
◊ GetSnapSpacing 450
◊ GetString 450
◊ GetSubEntity 451
◊ GetSubSelection 451
◊ GetText 452
◊ GetTextHeight 452
◊ GetTextRotation 453
◊ GetTextStyle 453
◊ GetUCSMatrix 453
◊ GetVariable 454
◊ GetWeight 454
◊ GetWidth 455
◊ GetWindowToPlot 455
◊ GetXData 455
◊ GetXRecordData 456
◊ HandleToObject 457
◊ Hide 457
◊ Highlight 457
◊ HitTest 458
◊ Import 458
◊ ImportProfile 459
◊ IndexOf 460
◊ InitializeUserInput 460

Метод *(прод.)*:
◊ InsertBlock 461
◊ InsertColumns 461
◊ InsertInMenuBar 462
◊ InsertLoopAt 462
◊ InsertMenuInMenuBar 463
◊ InsertRows 463
◊ Intersect 464
◊ IntersectWith 464, 878, 880
◊ IsConnected 465
◊ IsMergedCell 465
◊ IsRemoteFile 466
◊ IsURL 466
◊ Item 356, 467, 813
◊ LaunchBrowserDialog 467
◊ ListArx 468
◊ Load 468
◊ LoadArx 469
◊ LoadDVB 469
◊ LoadShapeFile 469
◊ MergeCells 470
◊ Mirror 326, 470
◊ Mirror3D 470
◊ Move 471
◊ MoveAbove 471
◊ MoveBelow 472
◊ MoveToBottom 472
◊ MoveToTop 473
◊ New 473
◊ NumCustomInfo 473
◊ ObjectIDToObject 474, 886
◊ Offset 474
◊ Open 184, 475
◊ PlotToDevice 475
◊ PlotToFile 475
◊ PolarPoint 476
◊ Prompt 476
◊ PurgeAll 477
◊ PurgeFitData 477
◊ PutRemoteFile 477
◊ Quit 477
◊ RealToString 478
◊ RecomputeTableBlock 478
◊ Refresh 478
◊ RefreshPlotDeviceInfo 479
◊ Regen 479
◊ Reload 480
◊ ReloadLabels 480
◊ Remove 481
◊ RemoveCustomByIndex 481
◊ RemoveCustomByKey 481
◊ RemoveEntry 481
◊ RemoveFromMenuBar 482
◊ RemoveItems 482
◊ RemoveMenuFromMenuBar 483
◊ Rename 483
◊ RenameProfile 483
◊ Replace 483
◊ ReselectSubRegion 484
◊ ResetBlock 484
◊ ResetProfile 484
◊ Restore 485
◊ Reverse 485
◊ Rotate 485
◊ Rotate3D 486
◊ RunMacro 486
◊ Save 487
◊ SaveAs 487
◊ ScaleEntity 488
◊ SectionSolid 488
◊ Select 489, 810
◊ SelectAtPoint 490
◊ SelectByPolygon 490
◊ SelectOnScreen 491, 810
◊ SelectSubRegion 492
◊ SendCommand 492, 793
◊ SendModelessOperationEnded 493
◊ SendModelessOperationStart 493
◊ SetAlignment 493
◊ SetAutoScale 494
◊ SetBackgroundColor 494
◊ SetBackgroundColorNone 495
◊ SetBitmaps 495, 874
◊ SetBlockAttributeValue 495
◊ SetBlockRotation 496
◊ SetBlockScale 496
◊ SetBlockTableRecordId 496
◊ SetBulge 497

◊ SetCellAlignment 497, 824
◊ SetCellBackgroundColor 498
◊ SetCellBackgroundColorNone 498
◊ SetCellContentColor 498
◊ SetCellGridColor 499
◊ SetCellGridLineWeight 499
◊ SetCellGridVisibility 500
◊ SetCellTextHeight 500
◊ SetCellTextStyle 500
◊ SetCellType 501
◊ SetColor 501
◊ SetColorBookColor 501
◊ SetColumnWidth 502, 824
◊ SetContentColor 502
◊ SetControlPoint 503
◊ SetCustomByIndex 503
◊ SetCustomByKey 503
◊ SetCustomScale 504
◊ SetDatabase 504
◊ SetFieldId 504
◊ SetFitPoint 505
◊ SetFont 505
◊ SetGridColor 505
◊ SetGridLineWeight 506
◊ SetGridSpacing 506
◊ SetGridVisibility 506
◊ SetInvisibleEdge 507
◊ SetLayoutsToPlot 507
◊ SetNames 508
◊ SetPattern 508
◊ SetProjectFilePath 508
◊ SetRelativeDrawOrder 509
◊ SetRGB 509
◊ SetRowHeight 509
◊ SetSnapSpacing 510
◊ SetSubSelection 510
◊ SetText 510, 824
◊ SetTextHeight 511, 824
◊ SetTextRotation 511
◊ SetTextStyle 511
◊ SetVariable 512
◊ SetView 512
◊ SetWeight 512
◊ SetWidth 513
◊ SetWindowToPlot 513
◊ SetXData 513
◊ SetXRecordData 514
◊ Show 515
◊ SliceSolid 515
◊ Split 516
◊ StartBatchMode 516
◊ StartUndoMark 517
◊ Subtract 517
◊ SwapOrder 517
◊ SyncModelView 518
◊ TransformBy 518
◊ TranslateCoordinates 518
◊ Union 519
◊ Unload 520
◊ UnloadArx 520
◊ UnloadDVB 521
◊ UnmergeCells 521
◊ Update 521
◊ UpdateEntry 521
◊ Wblock 522
◊ ZoomAll 522
◊ ZoomCenter 522
◊ ZoomExtents 523
◊ ZoomPickWindow 523
◊ ZoomPrevious 523
◊ ZoomScaled 523
◊ ZoomWindow 524

Н

Набор 121
◊ закрытие 122

О

Обработка ошибок 82
Объект:
◊ ACDBDICTIONARYWDFLT 147
◊ AcEdCommandStack 336
◊ Application 316, 324
◊ Application (Excel) 815
◊ Block 887
◊ DICTIONARY 147, 150
◊ Document 317
◊ IDPair 884

Объект *(прод.)*:
◊ Insert 889
◊ ModelSpace 325
◊ PaperSpace 325
◊ VL-CATCH-ALL-APPLY-ERROR 82, 868, 873
◊ VLO-VL 347
◊ XRECORD 152
Оператор 236
◊ boxed_column 259
◊ boxed_row 259
◊ column 257, 291
◊ row 257, 258, 291

П

Панель:
◊ добавление кнопки 870
◊ перемещение 870
◊ создание 870
◊ удаление 870
Параметр 355
Пароль 293
Переключатель 254
Переменная 8, 12
◊ $value 276
◊ $x 283, 285
◊ $y 285
◊ глобальная 182
◊ действия 264, 266, 283
◊ локальная 156, 180
◊ окружения 91
◊ окружения ACADHELP 312
◊ окружения TEMP 109
◊ окружения TMP 109
◊ рабочая 155, 181
Планшет 195
Полигрань 857, 858
Прерывание 81
Примитив:
◊ ACAD_TABLE 815
◊ ATTRIBUTE 859
◊ CIRCLE 854, 855
◊ INSERT 120, 859
◊ LINE 854
◊ LWPOLYLINE 115, 855, 856
◊ POLYLINE 114, 120, 856
◊ SEQEND 114, 859
◊ VERTEX 114, 857, 858
◊ VIEWPORT 117, 118, 143
◊ VLO-VL 347
◊ WIPEOUT 144
◊ восстановление 115
◊ выбор 128
◊ имя 114
◊ тип 116
Проект Visual LISP 214, 225
Пространство:
◊ внедокументное 13, 178
◊ имен 173
Пустой ввод 83

Р

Раздел:
◊ ACCELERATORS 295
◊ AUX1 295
◊ BUTTONS1 295
◊ HELPSTRINGS 295
◊ IMAGE 303
◊ POP0 304
◊ POP1 298
◊ POP499 298
◊ POP500 304
◊ SCREEN 296, 302
◊ TABLET 296
◊ TOOLBARS 303
Расширенные данные 139, 152, 212, 849, 881
Реактор 756, 846, 851, 853
◊ :VLR-AcDb-Reactor 761, 790
◊ :VLR-Command-Reactor 768
◊ :VLR-DeepClone-Reactor 768
◊ :VLR-DocManager-Reactor 762
◊ :VLR-DWG-Reactor 769
◊ :VLR-DXF-Reactor 770, 791
◊ :VLR-Editor-Reactor 765
◊ :VLR-Insert-Reactor 771
◊ :VLR-Linker-Reactor 763
◊ :VLR-Lisp-Reactor 772

◊ :VLR-Miscellaneous-Reactor 773
◊ :VLR-Mouse-Reactor 774
◊ :VLR-Object-Reactor 764, 786, 794
◊ :VLR-SysVar-Reactor 774
◊ :VLR-Toolbar-Reactor 775
◊ :VLR-Undo-Reactor 776
◊ :VLR-Wblock-Reactor 777
◊ :VLR-Window-Reactor 779
◊ :VLR-XREF-Reactor 780
◊ ARX-приложений 763
◊ DWG-операций 769
◊ DXF-операций 770
◊ LISP-операций 772
◊ базы рисунка 761
◊ внешних ссылок 780
◊ временный 787
◊ вставки блока 771
◊ действие 759
◊ диспетчера документов 762
◊ клонирования 768
◊ кнопок панелей 775
◊ команд 768, 792
◊ команды WBLOCK 777
◊ мыши 774, 793
◊ объектов 761, 764, 797
◊ окон 779
◊ отмены 776
◊ постоянный 787
◊ прочих событий 773
◊ редактирования 765
◊ системных переменных 774
◊ создание 760
◊ функция действия 759
Регистрация приложения 141
Режим:
◊ SNAP 450, 510
◊ многодокументный 230
Рекурсия 800
Ручки 130

С

Сборка мусора 169
Свойство:
◊ Action 527
◊ Active 528
◊ ActiveDimStyle 528
◊ ActiveDocument 528
◊ ActiveLayer 529
◊ ActiveLayout 529
◊ ActiveLinetype 529
◊ ActiveProfile 529
◊ ActivePViewport 530
◊ ActiveSelectionSet 530, 810
◊ ActiveSpace 530
◊ ActiveTextStyle 530
◊ ActiveUCS 531
◊ ActiveViewport 531
◊ ADCInsertUnitsDefaultSource 531
◊ ADCInsertUnitsDefaultTarget 532
◊ AffectsGraphics 532
◊ Algorithm 533
◊ Alignment 533
◊ AlignmentPointAcquisition 534
◊ AllowedValues 534
◊ AllowLongSymbolNames 535
◊ AltFontFile 535
◊ AltRoundDistance 535
◊ AltSuppressLeadingZeros 536
◊ AltSuppressTrailingZeros 536
◊ AltSuppressZeroFeet 536
◊ AltSuppressZeroInches 537
◊ AltTabletMenuFile 537
◊ AltTextPrefix 537
◊ AltTextSuffix 538
◊ AltTolerancePrecision 538
◊ AltToleranceSuppressLeadingZeros 538
◊ AltToleranceSuppressTrailingZeros 539
◊ AltToleranceSuppressZeroFeet 539
◊ AltToleranceSuppressZeroInches 540
◊ AltUnits 540
◊ AltUnitsFormat 540
◊ AltUnitsPrecision 541
◊ AltUnitsScale 541
◊ Angle 541
◊ AngleFormat 542
◊ AngleVertex 526, 542
◊ Annotation 542
◊ Application 543
◊ ArcEndParam 543

Свойство *(прод.)*:
◊ ArcLength 543
◊ ArcPoint 544
◊ ArcSmoothness 544
◊ ArcStartParam 544
◊ Area 545
◊ Arrowhead1Block 545
◊ Arrowhead1Type 545
◊ Arrowhead2Block 546
◊ Arrowhead2Type 547
◊ ArrowheadBlock 547
◊ ArrowheadSize 547
◊ ArrowheadType 548
◊ AssociativeHatch 548
◊ AttachmentPoint 548
◊ Author 549
◊ AutoAudit 549
◊ AutomaticPlotLog 549
◊ AutoSaveInterval 550
◊ AutoSavePath 550
◊ AutoSnapAperture 550
◊ AutoSnapApertureSize 551
◊ AutoSnapMagnet 551
◊ AutoSnapMarker 551
◊ AutoSnapMarkerColor 552
◊ AutoSnapMarkerSize 552
◊ AutoSnapToolTip 553
◊ AutoTrackingVecColor 553
◊ AutoTrackTooltip 553
◊ BackgroundFill 554
◊ Backward 554
◊ BasePoint 554
◊ BatchPlotProgress 555
◊ BeepOnError 555
◊ BigFontFile 555
◊ BitFlags 556
◊ Block 556
◊ Blocks 556
◊ BlockScaling 557
◊ Blue 557
◊ BookName 557
◊ Brightness 557
◊ CanonicalMediaName 558
◊ Caption 558
◊ Catalog 558
◊ CategoryName 559
◊ Center 559
◊ CenterMarkSize 559
◊ CenterPlot 560
◊ CenterPoint 560
◊ CenterType 560
◊ Centroid 561
◊ Check 561
◊ ChordPoint 562
◊ Circumference 562
◊ Clipped 562
◊ ClippingEnabled 563
◊ Closed 563
◊ Color 563, 793
◊ ColorBookPath 564
◊ ColorIndex 564
◊ ColorMethod 564
◊ ColorName 565
◊ Columns 565
◊ ColumnSpacing 565
◊ ColumnWidth 566
◊ Comment 566
◊ Comments 566
◊ ConfigFile 566
◊ ConfigName 567
◊ Constant 567
◊ ConstantWidth 567
◊ ContinuousPlotLog 567
◊ ContourLinesPerSurface 568
◊ Contrast 568
◊ ControlPoints 568
◊ Coordinate 569
◊ Coordinates 569
◊ Count 356, 570
◊ CreateBackup 570
◊ CursorSize 570
◊ CustomDictionary 571
◊ CustomScale 571
◊ Database 571
◊ DataSource 571
◊ DataSourceLocation 572
◊ DbConnect 572
◊ DecimalSeparator 572

◊ DefaultInternetURL 573
◊ DefaultOutputDevice 573
◊ DefaultPlotStyleForLayer 573
◊ DefaultPlotStyleForObjects 574
◊ DefaultPlotStyleTable 574
◊ DefaultPlotToFilePath 574
◊ DefinedSize 575
◊ Degree 575
◊ Delta 575
◊ DemandLoadARXApp 576
◊ Description 576
◊ Diameter 576
◊ Dictionaries 577
◊ DimensionLineColor 577
◊ DimensionLineExtend 577
◊ DimensionLineWeight 578
◊ DimLine1Suppress 579
◊ DimLine2Suppress 579
◊ DimLineInside 579
◊ DimLineSuppress 580
◊ DimStyles 580
◊ Direction 580
◊ DirectionVector 581
◊ Display 581
◊ DisplayableAttributes 582
◊ DisplayGrips 582
◊ DisplayGripsWithinBlocks 582
◊ DisplayLayoutTabs 583
◊ DisplayLocked 583
◊ DisplayOLEScale 583
◊ DisplayScreenMenu 584
◊ DisplayScrollBars 584
◊ DisplaySilhouette 584
◊ DisplayValues 585
◊ DockedVisibleLines 585
◊ DockStatus 586
◊ Document 586
◊ Documents 586
◊ Drafting 587
◊ DrawingDirection 587
◊ DriversPath 587
◊ EffectiveName 588
◊ Elevation 588
◊ ElevationModelSpace 588
◊ ElevationPaperSpace 589
◊ Enable 589
◊ EnableStartupDialog 589
◊ EndAngle 590
◊ EndParameter 590
◊ EndPoint 327, 590
◊ EndSubMenuLevel 591
◊ EndTangent 591
◊ EntityColor 591
◊ Explodable 592
◊ ExtensionLineColor 592
◊ ExtensionLineExtend 592
◊ ExtensionLineOffset 593
◊ ExtensionLineWeight 593
◊ ExtLine1EndPoint 593
◊ ExtLine1Point 594
◊ ExtLine1StartPoint 594
◊ ExtLine1Suppress 594
◊ ExtLine2EndPoint 595
◊ ExtLine2Point 595
◊ ExtLine2StartPoint 595
◊ ExtLine2Suppress 596
◊ Fade 596
◊ Feature 596
◊ FieldLength 596
◊ FieldName 597
◊ FileDependencies 597
◊ FileName 597
◊ Files 598
◊ FileSize 598
◊ FingerprintGuid 598
◊ Fit 598
◊ FitPoints 599
◊ FitTolerance 599
◊ FloatingRows 600
◊ FlowDirection 600
◊ Flyout 600
◊ FontFile 601
◊ FontFileMap 601
◊ ForceLineInside 601
◊ Format 602
◊ FoundPath 602
◊ FractionFormat 602
◊ Freeze 603

Свойство *(прод.)*:
◊ FullCRCValidation 603
◊ FullFileName 604
◊ FullName 604
◊ FullScreenTrackingVector 605
◊ GradientAngle 605
◊ GradientCentered 605
◊ GradientColor1 606
◊ GradientColor2 606
◊ GradientName 607
◊ GraphicsWinLayoutBackgrndColor 607
◊ GraphicsWinModelBackgrndColor 608
◊ Green 608
◊ GridOn 608
◊ GripColorSelected 609
◊ GripColorUnselected 609
◊ GripSize 610
◊ Groups 610
◊ Handle 610
◊ HasAttributes 610
◊ HasExtensionDictionary 145, 344, 611
◊ HasLeader 611
◊ HasSheetView 611
◊ HasSubSelection 612
◊ HasVpAssociation 612
◊ HatchObjectType 612
◊ HatchStyle 613
◊ HeaderSuppressed 613
◊ Height 613
◊ HelpFilePath 614
◊ HelpString 358, 614
◊ HistoryLines 614
◊ HorizontalTextPosition 615
◊ HorzCellMargin 615
◊ HWND 615
◊ HyperlinkBase 616
◊ HyperlinkDisplayCursor 616
◊ HyperlinkDisplayTooltip 616
◊ Hyperlinks 617
◊ ImageFile 617
◊ ImageFrameHighlight 617
◊ ImageHeight 617
◊ ImageVisibility 618
◊ ImageWidth 618
◊ IncrementalSavePercent 619
◊ Index 619
◊ InsertionPoint 619
◊ InsUnits 620
◊ InsUnitsFactor 620
◊ Invisible 620
◊ IsCloned 620, 886
◊ IsDynamicBlock 621
◊ IsLayout 621
◊ IsModified 622
◊ ISOPenWidth 622
◊ IsOwnerXlated 623
◊ IsPartial 623
◊ IsPeriodic 624
◊ IsPlanar 624
◊ IsPrimary 624
◊ IsQuiescent 625
◊ IsRational 625
◊ Issuer 625
◊ IsXRef 626
◊ JogAngle 626
◊ JogLocation 626
◊ Justification 627
◊ Key 627, 886
◊ KeyboardAccelerator 628
◊ KeyboardPriority 628
◊ KeyDescriptions 628
◊ KeyLength 629
◊ KeyValues 629
◊ Keywords 629
◊ Knots 630
◊ Label 630
◊ LabelBlockId 631
◊ LabelTemplate 631
◊ LargeButtons 632
◊ LastHeight 632
◊ LastSavedBy 632
◊ Layer 633
◊ LayerOn 633
◊ Layers 633
◊ LayerState 633
◊ Layout 634
◊ LayoutCreateViewport 634
◊ LayoutCrosshairColor 634

◊ LayoutDisplayMargins 635
◊ LayoutDisplayPaper 635
◊ LayoutDisplayPaperShadow 635
◊ LayoutId 636
◊ Layouts 636
◊ LayoutShowPlotSetup 636
◊ Leader1Point 637
◊ Leader2Point 637
◊ LeaderAttachment 637
◊ LeaderLength 638
◊ LeaderObjectId 638
◊ LeaderOffset 638
◊ Left 639
◊ Length 639
◊ LensLength 639
◊ Limits 640
◊ LinearScaleFactor 640
◊ LineSpacingDistance 641
◊ LineSpacingFactor 641
◊ LineSpacingStyle 641
◊ Linetype 642
◊ LinetypeGeneration 642
◊ Linetypes 642
◊ LinetypeScale 642
◊ LineWeight 643
◊ LineWeightDisplay 643
◊ Link 643
◊ LinkTemplate 644
◊ LinkType 644
◊ LoadAcadLspInAllDocuments 645
◊ LocaleId 645
◊ Lock 645
◊ LockAspectRatio 646
◊ LockPosition 646
◊ LogFileOn 646
◊ LogFilePath 647
◊ LowerLeftCorner 647
◊ Macro 647
◊ MainDictionary 648
◊ MajorAxis 648
◊ MajorRadius 648
◊ Mask 649
◊ MaxActiveViewports 650
◊ MaxAutoCADWindow 650
◊ MClose 650
◊ MDensity 651
◊ Measurement 651
◊ MenuBar 651
◊ MenuFile 651
◊ MenuFileName 652
◊ MenuGroups 652
◊ Menus 356, 652
◊ MinimumTableHeight 653
◊ MinimumTableWidth 653
◊ MinorAxis 653
◊ MinorRadius 653
◊ MLineScale 654
◊ Mode 654
◊ ModelCrosshairColor 655
◊ ModelSpace 655
◊ ModelType 655
◊ ModelView 656
◊ MomentOfInertia 656
◊ MRUNumber 656
◊ MSpace 657
◊ MTextObjectId 657
◊ MVertexCount 657
◊ Name 658
◊ NameNoMnemonic 658
◊ NClose 658
◊ NDensity 659
◊ Normal 659
◊ NumberOfControlPoints 660
◊ NumberOfCopies 660
◊ NumberOfFaces 660
◊ NumberOfFitPoints 660
◊ NumberOfLoops 661
◊ NumberOfVertices 661
◊ NumericScale 661
◊ NVertexCount 662
◊ ObjectID 662
◊ ObjectName 662
◊ ObjectSnapMode 662
◊ ObjectSortByPlotting 663
◊ ObjectSortByPSOutput 663
◊ ObjectSortByRedraws 663
◊ ObjectSortByRegens 664
◊ ObjectSortBySelection 664

Свойство *(прод.)*:
◊ ObjectSortBySnap 664
◊ ObliqueAngle 665
◊ OleItemType 665
◊ OLELaunch 666
◊ OlePlotQuality 666
◊ OLEQuality 666
◊ OleSourceApp 667
◊ OnMenuBar 667
◊ OpenSave 668
◊ Origin 668
◊ OrthoOn 668
◊ Output 668
◊ OverrideCenter 669
◊ OwnerID 669
◊ PageSetupOverridesTemplateFile 669
◊ PaperSpace 670
◊ PaperUnits 670
◊ Parent 670
◊ Password 671
◊ Path 671, 887
◊ PatternAngle 671
◊ PatternDouble 671
◊ PatternName 672
◊ PatternScale 672
◊ PatternSpace 673
◊ PatternType 673
◊ Perimeter 673
◊ PickAdd 674
◊ PickAuto 674
◊ PickBoxSize 674
◊ PickDrag 675
◊ PickFirst 675
◊ PickfirstSelectionSet 675
◊ PickGroup 675
◊ Plot 676
◊ PlotConfigurations 676
◊ PlotHidden 676
◊ PlotLegacy 677
◊ PlotLogFilePath 677
◊ PlotOrigin 677
◊ PlotPolicy 678
◊ PlotRotation 678
◊ PlotStyleName 678
◊ Plottable 679
◊ PlotType 679
◊ PlotViewportBorders 679
◊ PlotViewportsFirst 680
◊ PlotWithLineweights 680
◊ PlotWithPlotStyles 680
◊ PolarTrackingVector 681
◊ PostScriptPrologFile 681
◊ Precision 682
◊ Preferences 682
◊ Preset 682
◊ PrimaryUnitsPrecision 683
◊ PrincipalDirections 683
◊ PrincipalMoments 684
◊ PrinterConfigPath 684
◊ PrinterDescPath 684
◊ PrinterPaperSizeAlert 684
◊ PrinterSpoolAlert 685
◊ PrinterStyleSheetPath 685
◊ PrintFile 686
◊ PrintSpoolerPath 686
◊ PrintSpoolExecutable 686
◊ ProductOfInertia 686
◊ Profiles 687
◊ PromptString 687
◊ PropertyName 687
◊ ProviderName 688
◊ ProviderType 688
◊ ProxyImage 688
◊ QNewTemplateFile 689
◊ QuietErrorMode 689
◊ RadiiOfGyration 689
◊ Radius 690
◊ RadiusRatio 690
◊ ReadOnly 690
◊ Red 690
◊ ReferenceCount 691
◊ RegenerateTableSuppressed 691
◊ RegisteredApplications 691
◊ RenderSmoothness 691
◊ RevisionNumber 692
◊ Rotation 692
◊ RoundDistance 692
◊ RowHeight 693

◊ Rows 693
◊ RowSpacing 693
◊ SaveAsType 693
◊ Saved 694
◊ SavePreviewThumbnail 695
◊ ScaleFactor 695
◊ ScaleHeight 695
◊ ScaleLineweights 696
◊ ScaleWidth 696
◊ Schema 696
◊ SCMCommandMode 697
◊ SCMDefaultMode 697
◊ SCMEditMode 697
◊ SCMTimeMode 698
◊ SCMTimeValue 698
◊ SecondPoint 698
◊ SegmentPerPolyline 699
◊ Selection 699
◊ SelectionSets 699
◊ SerialNumber 699
◊ ShadePlot 700
◊ SheetView 700
◊ ShortcutMenu 700
◊ ShortCutMenuDisplay 701
◊ Show 701
◊ ShowPlotStyles 701
◊ ShowProxyDialogBox 702
◊ ShowRasterImage 702
◊ ShowRotation 702
◊ ShowWarningMessages 703
◊ SingleDocumentMode 703
◊ SnapBasePoint 704
◊ SnapOn 704
◊ SnapRotationAngle 704
◊ SolidFill 705
◊ StandardScale 705
◊ StartAngle 706
◊ StartParameter 707
◊ StartPoint 327, 707
◊ StartTangent 707
◊ StatusId 707
◊ StoreSQLIndex 708
◊ StyleName 708
◊ StyleSheet 709
◊ Subject 709
◊ SubMenu 709
◊ SummaryInfo 709
◊ SupportPath 710
◊ SuppressLeadingZeros 710
◊ SuppressTrailingZeros 710
◊ SuppressZeroFeet 711
◊ SuppressZeroInches 711
◊ SymbolPosition 711
◊ System 712
◊ Table 712
◊ TablesReadOnly 712
◊ TableStyleOverrides 713
◊ TabOrder 713
◊ TagString 713
◊ Target 714
◊ TempFileExtension 714
◊ TempFilePath 714
◊ TemplateDwgPath 715
◊ TempXrefPath 715
◊ Text 715
◊ TextAlignmentPoint 716
◊ TextAttachment 716
◊ TextColor 716
◊ TextEditor 717
◊ TextFont 717
◊ TextFontSize 717
◊ TextFontStyle 717
◊ TextFrameDisplay 718
◊ TextGap 718
◊ TextGenerationFlag 718
◊ TextHeight 719
◊ TextInside 719
◊ TextInsideAlign 720
◊ TextMovement 720
◊ TextOutsideAlign 721
◊ TextOverride 721
◊ TextPosition 721
◊ TextPrecision 722
◊ TextPrefix 722
◊ TextRotation 722
◊ TextString 723
◊ TextStyle 723
◊ TextStyles 723

Свойство *(прод.)*:
◊ TextSuffix 723
◊ TextureMapPath 724
◊ TextWinBackgrndColor 724
◊ TextWinTextColor 724
◊ Thickness 725
◊ TimeServer 725
◊ TimeStamp 725
◊ TipOffset 726
◊ Title 726
◊ TitleSuppressed 726
◊ ToleranceDisplay 727
◊ ToleranceHeightScale 727
◊ ToleranceJustification 728
◊ ToleranceLowerLimit 728
◊ TolerancePrecision 728
◊ ToleranceSuppressLeadingZeros 729
◊ ToleranceSuppressTrailingZeros 729
◊ ToleranceSuppressZeroFeet 729
◊ ToleranceSuppressZeroInches 730
◊ ToleranceUpperLimit 730
◊ Toolbars 730
◊ ToolPalettePath 731
◊ Top 731
◊ TotalAngle 731
◊ TranslateIDs 732
◊ Transparency 732
◊ TrueColor 732, 793
◊ TrueColorImages 733
◊ TwistAngle 733
◊ Type 733
◊ UCSIconAtOrigin 735
◊ UCSIconOn 736
◊ UCSPerViewport 736
◊ Units 736
◊ UnitsFormat 736
◊ UnitsType 737
◊ Updatable 737
◊ UpperRightCorner 738
◊ UpsideDown 738
◊ URL 738
◊ URLDescription 739
◊ URLNamedLocation 739
◊ Used 739
◊ UseLastPlotSettings 739
◊ User 740
◊ UserCoordinateSystems 740
◊ UseStandardScale 740
◊ Utility 741
◊ Value 741, 886
◊ VBE 741
◊ Verify 742
◊ Version 742
◊ VersionGuid 742
◊ VertCellMargin 742
◊ VerticalTextPosition 743
◊ ViewportDefault 743
◊ ViewportOn 744
◊ Viewports 744
◊ Views 744
◊ ViewToPlot 744
◊ VisibilityEdge1 745
◊ VisibilityEdge2 745
◊ VisibilityEdge3 745
◊ VisibilityEdge4 745
◊ Visible 746
◊ Volume 746
◊ Weights 746
◊ Width 747
◊ WindowLeft 747
◊ WindowState 747
◊ WindowTitle 748
◊ WindowTop 748
◊ WorkspacePath 748
◊ XEffectiveScaleFactor 748
◊ XRefDatabase 749
◊ XrefDemandLoad 749
◊ XRefEdit 749
◊ XRefFadeIntensity 750
◊ XRefLayerVisibility 750
◊ XScaleFactor 750
◊ XVector 750
◊ YEffectiveScaleFactor 751
◊ YScaleFactor 751
◊ YVector 751
◊ ZEffectiveScaleFactor 752
◊ ZScaleFactor 752

Секция:
◊ ENTITIES 851
◊ TABLES 852
Семейство:
◊ Documents 184, 317, 324
◊ Layers 344
◊ MenuBar 317
◊ MenuGroups 317, 873
◊ ModelSpace 326
◊ PopupMenus 317, 356
◊ Preferences 317
◊ Toolbars 317, 873
Сеть многогранная 857
Символ:
◊ LISP 8, 9
◊ nil 12
◊ pause 12
◊ pi 12
◊ T 12
◊ зарезервированный 12
◊ предопределенный 12
Система координат:
◊ мировая 53, 54
◊ объекта 54
◊ текущая 53
◊ экранная 53
◊ экранная листа 53
Системная переменная:
◊ _TOOLPALETTEPATH 731
◊ ACADLSPASDOC 201, 645
◊ AFLAGS 620, 654
◊ ANGBASE 52, 88, 418
◊ ANGDIR 88
◊ APBOX 550
◊ APERTURE 186, 551
◊ AUNITS 51, 52
◊ AUPREC 52
◊ AUTOSNAP 551, 553
◊ CECOLOR 90, 91
◊ CURSORSIZE 570
◊ CVPORT 137
◊ CVPORTS 136
◊ DBMOD 196
◊ DEFLPLSTYLE 573
◊ DEFPLSTYLE 574
◊ DIMADEC 722
◊ DIMALT 540
◊ DIMALTD 541
◊ DIMALTF 541
◊ DIMALTRND 535
◊ DIMALTTZ 538, 539, 540
◊ DIMALTU 540
◊ DIMALTZ 536
◊ DIMALTZ 536, 537
◊ DIMAPOST 538
◊ DIMASZ 547
◊ DIMATFIT 598
◊ DIMAUNIT 542
◊ DIMBLK1 545
◊ DIMBLK2 546, 547
◊ DIMCEN 559, 560
◊ DIMCLRD 577
◊ DIMCLRE 592
◊ DIMCLRT 716
◊ DIMDEC 683
◊ DIMDLE 577
◊ DIMDSEP 573
◊ DIMEXE 592
◊ DIMEXO 593
◊ DIMFRAC 602
◊ DIMGAP 718
◊ DIMJUST 615
◊ DIMLDRBLK 547, 548
◊ DIMLFAC 640
◊ DIMLUNIT 737
◊ DIMLWD 578
◊ DIMLWE 593
◊ DIMPOST 722, 723
◊ DIMRND 692
◊ DIMSCALE 695
◊ DIMSD1 579
◊ DIMSD2 579, 580
◊ DIMSE1 594
◊ DIMSE2 596
◊ DIMSOXD 579
◊ DIMTAD 743
◊ DIMTDEC 538, 728
◊ DIMTFAC 727

Системная переменная *(прод.)*:
◊ DIMTIH 720
◊ DIMTIX 719
◊ DIMTM 728
◊ DIMTMOVE 720
◊ DIMTOFL 601
◊ DIMTOH 721
◊ DIMTOL 727
◊ DIMTOLJ 728
◊ DIMTP 730
◊ DIMTXSTY 723
◊ DIMTXT 719
◊ DIMTZIN 729, 730
◊ DIMZIN 50, 710, 711
◊ DISPSILH 584
◊ DRAWORDERCTL 435, 448
◊ ELEVATION 54
◊ EXTNAMES 138, 535
◊ FACETRES 691
◊ FILLMODE 705
◊ GRIPBLOCK 582
◊ GRIPCOLOR 609
◊ GRIPHOT 609
◊ GRIPS 582
◊ GRIPSIZE 610
◊ HPANG 671
◊ HPDOUBLE 671
◊ HPNAME 672
◊ HPSCALE 672
◊ HPSPACE 673
◊ IMAGEHLT 617
◊ INSUNITSDEFSOURCE 531
◊ INSUNITSDEFTARGET 532
◊ ISAVEBAK 570
◊ ISAVEPERCENT 619
◊ ISOLINES 568
◊ LIMCHECK 83
◊ LIMMAX 640
◊ LIMMIN 640
◊ LOGFILEMODE 646
◊ LOGFILEPATH 647
◊ LUNITS 49, 52
◊ LUPREC 49
◊ LWDEFAULT 643
◊ MAXACTVP 650
◊ MODEMACRO 188
◊ OLEQUALITY 666
◊ OLESTARTUP 666
◊ OSMODE 91
◊ OSNAPCOORD 628
◊ PICKADD 674
◊ PICKAUTO 674
◊ PICKBOX 674
◊ PICKDRAG 675
◊ PICKFIRST 675
◊ POPUPS 83
◊ PROXYNOTICE 702
◊ PSTYLEPOLICY 678
◊ QTEXTMODE 718
◊ RASTERPREVIEW 695
◊ RTDISPLAY 702
◊ SAVEFILEPATH 550
◊ SAVETIME 550
◊ SCREENBOXES 188
◊ SDI 184, 230, 703
◊ SHORTCUTMENU 697, 698, 701
◊ SHOWLAYERUSAGE 417
◊ SNAPANG 704
◊ SNAPBASE 704
◊ SNAPMODE 704
◊ SORTENTS 663, 664
◊ SPLINESEGS 699
◊ TABMODE 195
◊ TILEMODE 44, 530
◊ TRACKPATH 605, 681
◊ UCSICON 735, 736
◊ UCSXDIR 750
◊ UCSYDIR 751
◊ UNITMODE 49, 52
◊ VISRETAIN 750
◊ XEDIT 749
◊ XFADECTL 750
◊ XLOADCTL 749
Скользящая шкала 256
Словарь 118, 144
◊ ACAD_LAYOUT 147, 148
◊ ACAD_MLINESTYLE 147
◊ ACAD_PLOTSTYLENAME 147

◊ extension dictionary 344
◊ корневой 145
◊ расширения объекта 344, 860
◊ тип 147
Слой 116
◊ создание 118
Событие 753
◊ :vlr-abortDxfIn 791
◊ :vlr-abortDxfOut 791
◊ :vlr-beginDwgOpen 788
◊ :vlr-commandEnded 788, 792
◊ :vlr-commandWillStart 788
◊ :vlr-endDwgOpen 788
◊ :vlr-modified 794
◊ :vlr-objectAppended 790
◊ :vlr-objectErased 790
Совместимость приложений 183
Сообщение об ошибке 55
Спецификация 815
Список 6, 7, 253
◊ раскрывающийся 254
Справочная система 309
Стек 196
Стиль текстовый 846
Структура файлов формата DXF 850

Т

Таблица символьная 133, 852
Технология COM 815
Тип:
◊ VARIANT 823
◊ линий 846
Тип линий 117
◊ масштаб 117, 847
Типы примитивов 113, 852
Тихий выход 101, 102
Точечная пара 11, 66, 70
Точечный список 11, 71
Точность отображения 8
Трассировка 166

Ф

Файл:
◊ acad.lsp 200
◊ acad.unt 55
◊ acad2006.cfg 93
◊ acad2006.lsp 200
◊ acad2006doc.lsp 201
◊ acaddoc.lsp 201
◊ acadinfo.lsp 826
◊ base.dcl 246
◊ slidelib.exe 282
◊ адаптации 294
◊ компилированного меню 294
◊ ресурсов адаптации 294
◊ ресурсов меню 294
◊ текстовой части меню 294
◊ шаблона меню 294
Фильтр слоев 344
◊ групповой 346
◊ по свойствам 346
Флажок 257
Функции:
◊ булевы 42
◊ перечень 826
◊ преобразования данных 47
Функция:
◊ * 18
◊ *error* 163
◊ / 19
◊ /= 22
◊ ~ 20
◊ + 17
◊ < 22
◊ <= 23
◊ = 21
◊ > 23
◊ >= 24
◊ 1– 20
◊ 1+ 19
◊ abs 20
◊ acad_colordlg 96
◊ acad_helpdlg 314
◊ acad_strlsort 75, 166
◊ acad_truecolorcli 99
◊ acad_truecolordlg 97, 285
◊ acad-pop-dbmod 196
◊ acad-push-dbmod 196
◊ acdimenableupdate 196
◊ acet-layerp-mark 197

Функция *(прод.)*:
◊ acet-layerp-mode 197
◊ acet-laytrans 198
◊ acet-ms-to-ps 198
◊ acet-ps-to-ms 199
◊ action_tile 243, 251, 263, 282, 291
◊ add_list 253, 254, 273, 281
◊ alert 103, 264, 336, 791
◊ alloc 168
◊ and 24
◊ angle 37
◊ angtof 51, 52
◊ angtos 51, 52
◊ append 67, 142
◊ apply 72, 81, 155
◊ arx 171
◊ arxload 171
◊ arxunload 171
◊ ascii 57, 103
◊ assoc 72, 116
◊ atan 39
◊ atof 50, 51
◊ atoi 12, 50
◊ atom 31
◊ atoms-family 154, 165
◊ autoarxload 172
◊ autoload 172
◊ boole 42
◊ boundp 32
◊ caaaar 70
◊ caaar 69
◊ caadar 70
◊ caadr 69
◊ caar 69
◊ cadr 69
◊ car 68, 131
◊ cdar 69
◊ cddr 69
◊ cdr 69, 116
◊ chr 56
◊ client_data_tile 264, 266
◊ close 100, 112
◊ command 12, 14, 17, 194, 201, 263, 759
◊ cond 27
◊ cons 11, 70
◊ cos 39
◊ cvunit 55
◊ defun 13, 155, 315
◊ defun-q 159
◊ defun-q-list-ref 160
◊ defun-q-list-set 161
◊ dictadd 149
◊ dictnext 147, 153
◊ dictremove 150
◊ dictrename 151
◊ dictsearch 146
◊ dimx_tile 279
◊ dimy_tile 279
◊ distance 38
◊ distof 52
◊ done_dialog 243, 244, 291
◊ end_image 253, 277, 281, 282
◊ end_list 253, 254, 273, 281
◊ entdel 115, 151
◊ entget 116, 123, 131, 136, 139, 759, 844, 853, 882
◊ entlast 53, 113
◊ entmake 54, 117, 141
◊ entmakex 118, 149
◊ entmod 118, 141, 142, 144, 759, 882
◊ entnext 53, 114
◊ entsel 53, 85, 115, 131, 788
◊ entupd 119
◊ eq 21, 32
◊ equal 21, 33
◊ eval 74, 75
◊ exit 162, 164
◊ exp 39
◊ expand 168
◊ expt 40
◊ fill_image 277, 282, 285, 863
◊ findfile 99, 108
◊ fix 12, 35
◊ float 12, 48
◊ foreach 74
◊ function 74, 77, 79—81, 161
◊ gc 167, 169, 183
◊ gcd 37

◊ get_attr 265, 284
◊ get_tile 265, 275
◊ getangle 88
◊ getcfg 93
◊ getcname 93
◊ getcorner 89
◊ getdist 87
◊ getenv 91
◊ getfiled 94
◊ getint 10, 86
◊ getkword 85, 788
◊ getorient 89
◊ getpoint 89, 788
◊ getreal 87
◊ getstring 85, 90
◊ getvar 90
◊ graphscr 187
◊ grclear 191
◊ grdraw 190
◊ grread 191
◊ grtext 187
◊ grvecs 190
◊ handent 119
◊ help 312, 313
◊ if 26
◊ initdia 194
◊ initget 83, 306, 824
◊ inters 43
◊ itoa 49
◊ lambda 73, 77, 79—81, 154
◊ last 75
◊ layoutlist 151
◊ length 76
◊ list 66
◊ listp 30
◊ load 6, 13, 162, 233
◊ load_dialog 242
◊ log 40
◊ logand 40, 43
◊ logior 41, 43
◊ lsh 41
◊ mapcar 73, 155, 161, 273
◊ max 36
◊ mem 167
◊ member 71
◊ menucmd 297
◊ menugroup 296
◊ min 37
◊ minusp 29
◊ mode_tile 264
◊ namedobjdict 145
◊ nentsel 53, 85, 120
◊ nentselp 85, 121, 191
◊ new_dialog 243, 263, 264, 291
◊ not 25
◊ nth 67, 131, 276
◊ null 25
◊ numberp 30, 868
◊ open 100, 112
◊ or 24
◊ osnap 185
◊ PL:ReadFile 803
◊ PL:String->List 802
◊ PL:String-Rep 801
◊ polar 44
◊ prin1 102, 111
◊ princ 101, 110, 112, 358
◊ print 102
◊ progn 26
◊ prompt 103
◊ quit 163
◊ quote 75
◊ read 57, 868
◊ read-char 102, 112
◊ read-line 100
◊ redraw 186, 190
◊ regapp 141
◊ rem 36
◊ repeat 35, 863
◊ reverse 68
◊ rtos 12, 49, 52
◊ s::startup 201
◊ set 47, 179
◊ set_tile 240, 252, 260, 265, 276
◊ setcfg 93
◊ setenv 92
◊ setfunhelp 315
◊ setq 8, 12

Функция *(прод.)*:
◊ setvar 91
◊ setview 136
◊ sin 38
◊ slide_image 278, 281, 282
◊ snvalid 138
◊ sqrt 38
◊ ssadd 126
◊ ssdel 126
◊ ssget 122, 131, 846
◊ ssgetfirst 131
◊ sslength 127, 131
◊ ssmemb 127
◊ ssname 53, 127, 131
◊ ssnamex 123, 128
◊ sssetfirst 130
◊ start_dialog 243, 244, 263, 291
◊ start_image 253, 276, 281, 282
◊ start_list 253, 254, 272, 281
◊ startapp 185, 313
◊ strcase 58
◊ strcat 59
◊ strlen 59
◊ subst 76
◊ substr 59
◊ tablet 195
◊ tblnext 134, 852
◊ tblobjname 135, 852
◊ tblsearch 134, 852
◊ term_dialog 243, 244
◊ terpri 104
◊ textbox 44
◊ textpage 187
◊ textscr 187
◊ trace 166
◊ trans 53
◊ type 10, 47, 868
◊ unload_dialog 242
◊ untrace 166, 167
◊ vector_image 278, 285
◊ ver 167
◊ vla-AddCircle 328
◊ vla-ArrayPolar 326
◊ vla-ArrayRectangular 326
◊ vl-acad-defun 164
◊ vl-acad-undefun 164
◊ vla-Copy 326
◊ vla-CopyObjects 884
◊ vla-get-ActiveDocument 324
◊ vla-get-area 324
◊ vla-get-color 324
◊ vla-get-Color 327
◊ vla-get-Documents 324
◊ vla-get-EndPoint 328
◊ vla-get-Layer 327
◊ vla-get-Length 796
◊ vla-get-Linetype 327
◊ vla-get-ModelSpace 325
◊ vla-get-PaperSpace 325
◊ vla-get-radius 324
◊ vla-get-StartPoint 328
◊ vla-Mirror 326
◊ vla-Move 794
◊ vla-put-Center 799
◊ vla-put-color 324
◊ vla-put-Color 328, 794, 799
◊ vla-put-Radius 799
◊ vla-put-StartPoint 759
◊ vl-arx-import 173
◊ vla-SendCommand 793
◊ vlax-3D-point 330
◊ vlax-add-cmd 335
◊ vlax-create-object 339, 815
◊ vlax-curve-getArea 351
◊ vlax-curve-getClosestPointTo 354
◊ vlax-curve-getClosestPointToProjection 354
◊ vlax-curve-getDistAtParam 353
◊ vlax-curve-getDistAtPoint 352
◊ vlax-curve-getEndParam 352
◊ vlax-curve-getEndPoint 351
◊ vlax-curve-getFirstDeriv 355
◊ vlax-curve-getParamAtDist 353
◊ vlax-curve-getParamAtPoint 352
◊ vlax-curve-getPointAtDist 351
◊ vlax-curve-getPointAtParam 353
◊ vlax-curve-getSecondDeriv 355
◊ vlax-curve-getStartParam 352

◊ vlax-curve-getStartPoint 351
◊ vlax-curve-isClosed 353
◊ vlax-curve-isPeriodic 354
◊ vlax-curve-isPlanar 354
◊ vlax-dump-object 324, 326, 333, 816
◊ vlax-ename->vla-object 327
◊ vlax-erased-p 338
◊ vlax-for 341, 814
◊ vlax-get 526
◊ vlax-get-acad-object 324
◊ vlax-get-object 340, 526, 815
◊ vlax-get-or-create-object 340, 815
◊ vlax-get-property 341, 350, 525
◊ vlax-import-type-library 340
◊ vlax-invoke 337, 361
◊ vlax-invoke-method 337, 341, 360
◊ vlax-ldata-delete 145, 343
◊ vlax-ldata-get 145, 343
◊ vlax-ldata-list 145, 343
◊ vlax-ldata-put 145, 342, 346
◊ vlax-ldata-test 145, 344
◊ vlax-make-safearray 331
◊ vlax-make-variant 328
◊ vlax-map-collection 341
◊ vlax-method-applicable-p 338
◊ vlax-object-released-p 338
◊ vlax-product-key 337
◊ vlax-property-available-p 350
◊ vlax-put 526
◊ vlax-put-property 341, 350, 525
◊ vlax-read-enabled-p 339
◊ vlax-release-object 339, 874
◊ vlax-remove-cmd 336
◊ vlax-safearray->list 349
◊ vlax-safearray-fill 332
◊ vlax-safearray-get-dim 348
◊ vlax-safearray-get-element 349
◊ vlax-safearray-get-l-bound 348, 880
◊ vlax-safearray-get-u-bound 348, 880
◊ vlax-safearray-put-element 332
◊ vlax-safearray-type 349
◊ vlax-tmatrix 349
◊ vlax-typeinfo-available-p 338
◊ vlax-variant-change-type 330
◊ vlax-variant-type 329
◊ vlax-variant-value 330, 823
◊ vlax-vla-object->ename 327, 346
◊ vlax-write-enabled-p 339
◊ vl-bb-ref 178, 230
◊ vl-bb-set 178, 230
◊ vl-catch-all-apply 81, 164, 799, 868, 873, 887
◊ vl-catch-all-error-message 81, 82, 164
◊ vl-catch-all-error-p 164
◊ vl-cmdf 16, 194, 201, 759
◊ vl-consp 31
◊ vl-directory-files 104
◊ vl-doc-export 173, 183, 230
◊ vl-doc-import 174, 183
◊ vl-doc-ref 179, 183
◊ vl-doc-set 179, 183, 230
◊ vl-every 73
◊ vl-exit-with-error 164, 165
◊ vl-exit-with-value 163, 164, 165
◊ vl-file-copy 105
◊ vl-file-delete 106
◊ vl-file-directory-p 106
◊ vl-filename-base 108
◊ vl-filename-directory 108
◊ vl-filename-extension 108
◊ vl-filename-mktemp 109
◊ vl-file-rename 107
◊ vl-file-size 107
◊ vl-file-systime 107
◊ vl-get-resource 177, 232
◊ vlisp-compile 162
◊ vl-list* 11, 71
◊ vl-list->string 62
◊ vl-list-exported-functions 159, 174
◊ vl-list-length 76
◊ vl-list-loaded-vlx 175
◊ vl-load-all 14, 180, 184
◊ vl-load-com 323
◊ vl-load-reactors 757
◊ vl-member-if 77
◊ vl-member-if-not 77
◊ vl-mkdir 110
◊ vl-position 78

Функция *(прод.)*:
◊ vl-prin1-to-string 111
◊ vl-princ-to-string 110
◊ vl-propagate 180, 184
◊ vlr-acdb-reactor 761
◊ vlr-add 785
◊ vlr-added-p 785
◊ vlr-beep-reaction 760, 790, 791, 875
◊ vlr-command-reactor 768
◊ vlr-current-reaction-name 785
◊ vlr-data 785, 789
◊ vlr-data-set 785
◊ vlr-deepclone-reactor 768
◊ vlr-docmanager-reactor 762
◊ vlr-dwg-reactor 769
◊ vlr-dxf-reactor 770
◊ vlr-editor-reactor 765
◊ vl-registry-delete 170
◊ vl-registry-descendents 170
◊ vl-registry-read 169
◊ vl-registry-write 169
◊ vl-remove 78
◊ vl-remove-if 78
◊ vl-remove-if-not 79
◊ vlr-insert-reactor 771
◊ vlr-linker-reactor 763
◊ vlr-lisp-reactor 772
◊ vlr-miscellaneous-reactor 773
◊ vlr-mouse-reactor 774
◊ vlr-notification 786
◊ vlr-object-reactor 764
◊ vlr-owner-add 786
◊ vlr-owner-remove 786
◊ vlr-owners 786
◊ vlr-pers 787
◊ vlr-pers-list 788
◊ vlr-pers-p 788
◊ vlr-pers-release 787
◊ vlr-reaction-names 759
◊ vlr-reactions 784
◊ vlr-reaction-set 783
◊ vlr-reactors 758
◊ vlr-remove 784, 790
◊ vlr-remove-all 784, 790
◊ vlr-set-notification 786
◊ vlr-sysvar-reactor 774
◊ vlr-toolbar-reactor 775
◊ vlr-trace-reaction 760
◊ vlr-type 759
◊ vlr-types 758
◊ vlr-undo-reactor 776
◊ vlr-wblock-reactor 777
◊ vlr-window-reactor 779
◊ vlr-xref-reactor 780
◊ vl-some 79
◊ vl-sort 80
◊ vl-sort-i 81
◊ vl-string->list 61
◊ vl-string-elt 62
◊ vl-string-left-trim 62
◊ vl-string-mismatch 63
◊ vl-string-position 64, 868
◊ vl-string-right-trim 63
◊ vl-string-search 64
◊ vl-string-subst 65
◊ vl-string-translate 65
◊ vl-string-trim 63
◊ vl-symbol-name 66
◊ vl-symbolp 32
◊ vl-symbol-value 47
◊ vl-unload-vlx 176
◊ vl-vbaload 177
◊ vl-vbarun 177
◊ vl-vlx-loaded-p 176
◊ vports 44
◊ wcmatch 60, 124, 210
◊ while 33, 824
◊ write-char 103, 112
◊ write-line 101, 112
◊ xdroom 143
◊ xdsize 143
◊ zerop 30
◊ арифметическая 17
◊ безымянная 154
◊ вычислительная 35
◊ действия 759, 790
◊ загрузка 157
◊ логическая 21

◊ создание 155
◊ тело 155
◊ экспортированная 174

Ц

Цвет 117
Центроид 561

Ч

Число:
◊ вещественное 10
◊ целое 10
Чтение значения глобальной переменной 224

Ш

Шифрование 293

Э

Элемент:
◊ boxed_radio_row 256
◊ button 251
◊ concatenation 260
◊ edit_box 252, 864, 867
◊ errtile 252
◊ image 252, 281, 863, 867
◊ image_button 252
◊ list_box 253, 272, 281
◊ ok_button 246
◊ ok_cancel 246
◊ ok_cancel_err 247
◊ ok_cancel_help 246
◊ ok_cancel_help_errtile 247
◊ ok_cancel_help_info 247
◊ ok_only 246
◊ paragraph 260
◊ popup_list 254, 272
◊ radio_button 255
◊ radio_column 255, 256
◊ radio_row 255
◊ slider 256
◊ spacer 261, 291
◊ spacer_0 261
◊ spacer_1 261, 291
◊ text 260
◊ text_part 260
◊ toggle 257

Я

Язык DCL 235