Акцент 7.4

язык программирования VBScript

руководство программиста

Донецк
Impact Ltd.
© 2001-2008

Редакция 7.4 31 марта 2008г.

Все права защищены. Никакая часть этой книги не может быть воспроизведена в любой форме или любыми средствами, электронными или механическими, включая фотографию, магнитную запись или иные средства копирования, или сохранения информации без письменного разрешения *Impact Ltd*.

Microsoft®, Microsoft Excel®, Windows®, Windows NT®, Win32® являются зарегистрированными товарными знаками Microsoft® Corporation. Visual Basic™, ActiveX™ являются товарными знаками Microsoft® Corporation. Названия других продуктов и компаний могут быть товарными знаками соответствующих компаний.

© Copyright 2001-2008 by *Impact Ltd*

Краткое содержание

Введение 13
Глава I. VBScript и Акцент 17
Глава II. Основы VBScript 23
Глава III. Инструкции VBScript 31
Глава IV. Встроенные функции 49
Глава V. Объекты Акцент 93
Глава VI. Отчеты 141
Глава VII. Формы 169
Глава VIII. Диалоги 195
Глава IX. Электронные таблицы 223
Глава X. Доступ к данным (DAO) 233
Глава XI. Доступ к данным (ADO) 247
Глава XII. Дополнительные сведения 263
Приложения 271
Алфавитный указатель 277

Содержание

Введение 13
Для кого эта книга 13
Обзор глав книги 14
Типографские соглашения 15
Описание синтаксиса 15
Формат описания функций, свойств и методов 16
Техническая поддержка 16

Глава I. VBScript и Акцент 17
Структура приложения Акцент 17
Расположение файлов 18
Директива #include 19
Где используется VBScript 19
Модули 20
Формы 20
Диалоги 20
Электронные таблицы 21
Модуль Workarea 21
Связь модулей, форм, диалогов и таблиц 22

Глава II. Основы VBScript 23
Формат программы 23
Комментарии 23
Литералы 24
Имена 24
Типы данных 25
Переменные и константы 26
Переменные 26
Константы 27
Объекты и коллекции 27
Объекты 27
Классы и экземпляры 27
Свойства методы и события 28
Коллекции 29

Глава III. Инструкции VBScript 31
Объявление переменных 32
Инструкция Dim 32

Массивы ... 33
Инструкция Erase ... 34
Инструкция ReDim ... 34
Присваивание ... 35
Инструкция Set ... 35
Операции и выражения ... 36
Арифметические операции ... 36
Операции сравнения ... 36
Логические операции ... 37
Операция конкатенации ... 37
Операция Is ... 38
Порядок выполнения операций ... 38
Управляющие инструкции ... 39
Инструкция If…Then…Else ... 39
Инструкция Select…Case ... 40
Инструкция For…Next ... 41
Инструкция Do…Loop ... 42
Инструкция For…Each ... 44
Инструкция While…Wend ... 45
Функции и процедуры ... 46
Инструкция Sub ... 46
Инструкция Function ... 47
Инструкция Call ... 47
Вспомогательные инструкции ... 48
Инструкция Execute ... 48

Глава IV. Встроенные функции ... 49

Преобразование данных ... 49
Функция CBool ... 50
Функция CByte ... 50
Функция CCur ... 50
Функция CDate ... 51
Функции CDbl, CSng ... 52
Функции CInt, CLng ... 52
Функция VarType ... 53
Математические функции ... 53
Функция Abs ... 54
Функция Sgn ... 54
Функции Fix и Int ... 54
Функция Round ... 55
Функция Round2 ... 55

Работа с массивами ... 56
Функция Array ... 56
Функции LBound, UBound ... 57
Функция IsArray ... 57
Работа с объектами ... 58
Функция IsObject ... 58
Функция CreateObject ... 58
Функция GetObject ... 59
Функция LoadPicture ... 60
Обработка строк ... 60
Функция Asc ... 62
Функция Chr ... 62
Функция CStr ... 62
Функции LTrim, RTrim, Trim ... 63
Функция Filter ... 64
Функции Hex и Oct ... 65
Функции InStr и InStrRev ... 65
Функция Join ... 66
Функции LCase и UCase ... 66
Функции Left и Right ... 67
Функция Len ... 67
Функция Mid ... 67
Функция Replace ... 68
Функция Space ... 68
Функция Split ... 69
Функция StrComp ... 69
Функция String ... 70
Функция StrReverse ... 70
Функции Акцент для обработки строк ... 71
Функция CharAt ... 71
Функция Spell ... 71
Функции SpellMoney, SpellMoney2 ... 72
Функция Token ... 73
Работа с датами ... 73
Функции Date, Time, Now ... 75
Функция DateAdd ... 75
Функция DateDiff ... 76
Функция DatePart ... 77
Функция DateSerial ... 77
Функция DateValue ... 78
Функции Day, Month, Year ... 78

Функция FormatDateTime ... 79
Функции Hour, Minute, Second ... 80
Функция IsDate ... 80
Функция TimeSerial ... 81
Функции Акцент для работы с датами ... 81
Функция FormatDate2 ... 81
Функции интерфейса пользователя ... 83
Функция MsgBox ... 83
Функция InputBox ... 85
Функция Ask ... 85
Ссылки на функции ... 86
Функция GetRef ... 86
Прочие функции Акцент ... 86
Функция IIF ... 87
Функция CheckNull ... 88
Функция DoEvent ... 88
Функция App ... 88
Функции Meter и WaitCursor ... 89
Функция QuickSort ... 89
Функция Workarea ... 91
Функция IsFileExists ... 91

Глава V. Объекты Акцент ... 93
Основные объекты ... 93
Объект Application ... 93
Многоуровневые коллекции ... 95
Пример работы с коллекцией ... 96
Объект Workarea ... 99
Объекты и база данных ... 104
Объекты Operation, Transaction, TransList ... 105
Объект Operation ... 106
Объект TransList ... 110
Объект Transaction ... 113
Проводки и налоговые отчеты ... 115
Связывание свойств с элементами управления ... 115
Работа с валютой ... 117
Объект Account и коллекция Accounts ... 118
Объект Agent и коллекция Agents ... 119
Объект Entity и коллекция Entities ... 120
Объект EntUnit и коллекция EntUnits ... 120
Объект Binder и коллекция Binders ... 121

Объект Template и коллекция Templates ... 121
Объект Param и коллекция Params ... 122
Объект Fact и коллекция Facts ... 123
Объект TaxReport и коллекция Tax Reports ... 124
Работа с шаблонами ... 125
Вспомогательные объекты ... 127
Объект Map ... 127
Объект Redirect ... 128
Объект Period ... 129
Объект PageSetup ... 129
Объекты Toolbar и ToolbarButton ... 130
Объект WaitCursor ... 132
Объект Meter ... 133
Объект PopupMenu ... 133
Объект FileDialog ... 134
Объект System ... 135
Объект WinAPI ... 136
Объект PopupToolTip ... 137
Объект DHTMLDialog ... 137
Объект DHTMLAssistant ... 139

Глава VI. Отчеты ... 141
Общие свойства и методы отчетов ... 142
Объекты отчетов ... 143
Отчет RepCommon ... 143
Отчет RepChess ... 144
Отчет RepBalance ... 145
Объект RepBalanceItem ... 146
Отчет RepCross ... 146
Отчет RepSubList ... 147
Объект RepSubListItem ... 149
Отчет RepDocList ... 149
Объект RepDocListItem ... 151
Отчет RepBankList ... 151
Объект RepBankListItem ... 152
Отчет RepCashBook ... 153
Объект RepCashBookItem ... 154
Отчет RepSynth ... 155
Отчет RepTurn ... 156
Отчет RepGLedger ... 157
Отчет RepJournal ... 158

Объект RepJournalItem 160
Отчет RepWizard 161
Объект RepWizardItem 164
Пример построения отчета `RepWizard` 165
Отчет RepTax 166

Глава VII. Формы 169
Объектная модель для форм 169
Создание формы 171
Объект Form 171
Объект FrmPage 175
Общие свойства и методы для элементов 176
Связывание данных 178
Элементы управления для форм 180
Элемент FrmLabel 180
Элемент FrmLine 181
Элемент FrmPicture 181
Элемент FrmButton 182
Элемент FrmCheck 182
Элемент FrmRadio 183
Элемент FrmRect 183
Элемент FrmEdit 183
Элемент FrmSelector 184
Элемент FrmDate 184
Элемент FrmComboBox 185
Элемент FrmGrid 185
Объект FrmGridRange 187
Элемент FrmScrollGrid 188
Объект FrmSGridColumn 192
Объект FrmSGridCell 193

Глава VIII. Диалоги 195
Объектная модель для диалогов 195
Создание диалога 196
Размеры диалога 196
Объект Dialog 197
Элементы управления диалогов 199
Общие свойства и методы для элементов 200
Элемент DlgLabel 202
Элемент DlgFrame 203
Элемент DlgGroupBox 203

Элемент DlgPicture ... 203
Элемент DlgButton ... 204
Элемент DlgCheck ... 205
Элемент DlgRadio ... 205
Элемент DlgEdit ... 205
Элемент DlgSelector ... 206
Элемент DlgDate ... 207
Элемент DlgListBox ... 207
Элемент DlgComboBox ... 208
Элемент DlgMeter ... 208
Элемент DlgGrid ... 209
Объект DlgGridRange ... 211
Элемент DlgTabStrip ... 213
Элемент DlgListView ... 214
Элемент DlgTreeView ... 218

Глава IX. Электронные таблицы ... 223
Объектная модель для таблиц ... 223
Формулы в таблицах ... 224
Создание таблицы ... 224
Объект ShtBook ... 225
Объект ShtSheet ... 228
Объект ShtRange ... 230

Глава X. Доступ к данным (DAO) ... 233
Объектная модель DAO ... 233
Доступ к DAO из Акцент ... 235
Константы DAO и файл DAO.INC ... 235
Объекты DAO ... 236
Объект DBEngine ... 236
Объект Workspace и коллекция Workspaces ... 237
Объект Error и коллекция Errors ... 237
Объект Database и коллекция Databases ... 237
Объект Recordset ... 239
Виды объектов Recordset ... 239
Объект Field и коллекция Fields ... 240
Доступ к Recordset ... 241
Навигация по объекту Recordset ... 241
Добавление, редактирование и удаление записей ... 242
Объект QueryDef ... 243
Доступ к объектам QueryDef ... 244

Объект Parameter и коллекция Parameters 245
Свойства и методы QueryDef..... 245

Глава XI. Доступ к данным (ADO) 247
Объектная модель ADO 248
Доступ к ADO из Акцент.....249
Константы ADO и файл ADO.INC 250
Динамические свойства 250
Объект Connection 251
Объект Command 254
Объект Recordset.....258
Объект Field и коллекция Fields.....261

Глава XII. Дополнительные сведения 263
Обработка ошибок и объект Err 263
Пример обработки ошибок 264
Классы в VBScript 264
Инструкция Class 264
Инструкции Property Get и Property Let 265
Инструкция Property Set 265
Работа с внешними приложениями.....265
HTML и Акцент 266
Доступ изVBScript к DOM 267
Доступ из HTML к объектной модели Акцент 267
DHTML-диалоги 268
Web-интерфейс 268
Встроенный браузер 269

Приложения 271
A. Список ключевых слов VBScript.....271
B. Стандартные объекты VBScript.....271
Объект Font 271
Объект Picture..... 272
Объект Dictionary..... 273
C. Стандартные Объекты WSH 274
Объекты FileSystemObject и TextStream 274
Пример работы с файловой системой..... 275

Алфавитный указатель 277

Введение

Во введении рассказывается о назначении языка *VBScript* и его месте в приложениях *Акцент*. Приводится краткое содержание глав книги.

Эта книга посвящена языку *Visual Basic Scripting Edition*™, разработанному корпорацией *Microsoft*® и используемому во многих программных продуктах на платформе *Windows*®. Язык построен на базе технологии *Active Scripting* и его можно использовать в программных продуктах сторонних производителей. Корпорация *Microsoft*® предоставляет лицензии на право такого использования языка.

Программные продукты семейства *Акцент 7.4*, разработанные фирмой *Импакт,* используют этот язык в качестве встроенного языка для настройки и разработки приложений.

Visual Basic Scripting Edition™ (часто называемый просто *VBScript)* это простой и мощный объектно-ориентированный язык программирования. Язык полностью поддерживает технологию *ActiveX*™ и за счет этого очень легко расширяется. Его можно использовать для непосредственного управления разнообразными приложениями, работающими в среде *Microsoft® Windows®.*

Язык очень прост для понимания и изучения. Если вы программировали на каком-либо из диалектов *Basic*, то *VBScript* покажется вам очень знакомым.

В этой книге *VBScript* описывается только применительно к программам семейства *Акцент*.

Для кого эта книга

Книга предназначена для специалистов, выполняющих разработку приложений в среде *Акцент 7.4*. Предполагается, что читатель знаком хотя бы с одним языком программирования и ориентируется в предметной области, связанной с бухгалтерским учетом и финансовыми приложениями. Максимальную пользу книга принесет тем специалистам, которые знают

какой-либо из диалектов языка *Basic*, например *Accent-Basic* версии 4.2, *Visual Basic* или *Visual Basic for Applications (VBA)*.

В книге не приводится базовая информация по программированию, хотя возможности *VBScript* описаны достаточно подробно. Основное внимание уделено особенностям применения языка в среде *Акцент 7.4*, описанию встроенных в систему объектов, их свойств, методов и событий.

Обзор глав книги

Книга состоит из введения, одиннадцати глав, приложений и алфавитного указателя.

В **первой главе** приводится описание структуры приложения *Акцент* и рассказывается об использовании языка *VBScript* в *Акцент*, а также приводится информация о структуре программных модулей.

Во **второй главе** дается краткое введение в язык, описываются типы данных, переменные и константы, а также дается краткое введение в объектно-ориентированное программирование.

Третья глава посвящена описанию инструкций языка *VBScript*. Дается краткое описание всех инструкций с примерами использования.

В **четвертой главе** описаны все встроенные функции, как стандартные (входящие в *VBScript*), так и специфические, добавленные в программные продукты *Акцент*.

Пятая глава посвящена объектам *Акцент*. В нее включено описание основных объектов системы, объектов для построения отчетов, а также информация о вспомогательных объектах.

Шестая глава описывает формы *Акцент*, элементы управления, используемые в формах и работу с ними.

В **седьмой главе** описаны диалоги *Акцент*, элементы управления диалогов и приводятся примеры их использования.

Восьмая глава посвящена электронным таблицам, используемым в программах *Акцент* для вывода отчетов.

В **девятой главе** приводится краткое описание объектной модели доступа к данным *Data Access Objects (DAO)*, используемой в настольных версиях программ семейства *Акцент*.

Десятая глава посвящена объектной модели *Active Data Objects (ADO)*, которая используется для работы с базой данных в редакциях «клиент-

сервер». К этим редакциям относятся версии *Акцент – MSDE* и *Акцент – SQL*.

В **одиннадцатой главе** описаны дополнительные сведения о *VBScript*. В частности работа с классами, обработка ошибок и прочая полезная информация.

В **приложениях** приводится список ключевых слов языка, краткий перечень возможных ошибок, описания некоторых стандартных объектов языка и расширения *Windows Scripting Host*.

Типографские соглашения

В книге используются следующие типографские соглашения.

- Программы и примеры набраны `равноширинным шрифтом`.
- Основные термины выделены *курсивом*.
- Имена файлов набраны в верхнем регистре. Например, FILE.TXT.
- Термины, представляющие собой ключевые слова языка, названия объектов, свойств, методов и функций набраны `равноширинным шрифтом`.
- Названия клавиш набраны прямым шрифтом. Если между наименованиями клавиш указан знак «плюс», то это означает, что нажимать их нужно вместе.
- Советы подсказки и примечания печатаются в рамках, залитых серым цветом.

Описание синтаксиса

При описании синтаксиса языка используются следующие символы:

Символ	Значение
[ ]	В квадратные скобки заключаются необязательные синтаксические элементы.
|	Вертикальной чертой разделяются элементы, из которых нужно выбрать только один.
()	В круглые скобки заключаются списки аргументов функций, свойств и методов а также индексы массивов.
…	Многоточие означает, что конструкция может повторяться несколько раз.

Формат описания функций, свойств и методов

При описании синтаксиса свойств, методов и встроенных функций используется следующий формат. Описание состоит из синтаксической диаграммы, описания аргументов и примера применения. Описание иллюстрируется следующим примером:

Function() – наименование функции

Краткое описание назначения функции и особенностей ее применения.

Синтаксис

```
Function(аргумент1,[аргумент2])
```

Аргументы

`аргумент1` – описание первого аргумента.
`аргумент2` – описание второго (необязательного) аргумента.

Пример

```
Здесь приводится пример использования функции в
программах
```

Техническая поддержка

Система постоянно развивается и улучшается. Поэтому, в книге могут не найти отражение самые последние возможности, добавленные в систему после того, как книга вышла из печати. Самые последние сведения приведены в электронной справочной системе, поставляемой вместе с программой. Чтобы быть постоянно в курсе событий, обращайтесь на `http://www.accent.ua`.

Книга и примеры в ней тщательно проверены. Однако мы не можем гарантировать полного отсутствия опечаток и неточностей. Если Вы нашли ошибку или неточность в этой книге, обратитесь непосредственно в фирму *Impact Ltd.*

- по телефону +(380)-062-389-81-40 («горячая линия»)
- по e-Mail `support@accent6.com`
- на WWW `http://www.accent.ua`
- по почте 83017, Украина, Донецк,
  ул. Белинского 24а.

Глава I.

VBScript и Акцент

В этой главе описана структура приложения *Акцент*, место языка VBScript в нем, рассказано о том, из чего оно состоит, и как части приложения взаимодействуют между собой. Кроме того, приведена информация о структуре модулей внутри приложения.

Структура приложения Акцент

Обычно в программах на *Visual Basic* существует понятие приложения. Все элементы такого приложения, такие как формы или модули компилируются одновременно (обычно сразу после загрузки документа-приложения) и существуют в таком виде все время, пока приложение выполняется. В этом случае все части приложения выполняются в едином глобальном контексте. В таких приложениях возможно использование глобальных переменных, из одного модуля можно использовать функции другого модуля и т.д. Фактически, модули в этом случае используются только для более удобного размещения подпрограмм в соответствии с их смыслом. Во время выполнения программы все модули загружены в память и доступны в любой момент.

В программах семейства *Акцент* используется несколько другой подход, скорее напоминающий работу сценариев на WWW-страницах. Каждый объект системы, такой, например, как форма или диалог, существует вполне независимо от других и выполняется только по специальной команде и в своем собственном контексте. Модули полностью независимы друг от друга.

Такой подход позволяет, в частности, устранить понятие «сборки приложения». Для того, чтобы подключить к программе новый модуль, форму или диалог, достаточно просто записать нужный файл в соответствующий каталог. После этого модуль сразу же можно использовать.

Однако, при таком подходе следует учитывать некоторые особенности программы. В частности, в программе *Акцент* невозможно объявить гло-

бальную переменную, так как непонятно, в каком контексте она должна храниться. Решение данной проблемы заключается в том, что в программе имеется возможность передавать данные из одной части приложения в другую с использованием специальных объектов. Более подробная информация об этом приведена в разделе, посвященном связи модулей между собой (стр. 22).

Расположение файлов

Способ, в соответствии с которым система ищет конкретные файлы, определяется специальным текстовым файлом. Имя этого файла совпадает с именем приложения, а расширением будет `ard`. Например, для приложения `accent74.exe` файл расположения будет `accent74.ard`, а для файла `accent74s.exe – accent74s.ard`.

Расположение файлов настраивается с помощью задания текстовых строк, определяющих расширение и путь к файлу. Сначала задается маска файла и после знака равенства – путь для поиска. Программа будет искать файлы, совпадающие с указанной маской, в указанных после знака равенства папках.

Например, если в тексте файла расположения встречаются следующие строки:

```
*.afm = c:\accent7.4\forms
*.afm = c:\accent7.4\forms\common
*.afm = c:\accent7.4\forms\myforms
```

то при поиске файлов форм (с расширением `afm`) будут просматриваться следующие папки, причем именно в указанном порядке.

```
c:\accent7.4\forms
c:\accent7.4\forms\common
c:\accent7.4\forms\myforms
```

Если в системе хранится полный путь к файлу, то вся информация о расположении игнорируется.

Для повышения гибкости управления расположением файлов предусмотрены два специализированных макроса:

- **`$(AppPath)`** – содержит путь к папке, из которой запущено приложение.
- **`$(DbPath)`** –содержит путь к папке, из которой загружена текущая база данных.

Например, если в тексте файла встречается следующая строка:

```
*.afm = $(AppPath)\forms
```

и приложение запущено из папки `c:\accent7.4`, то при поиске файлов форм будет просматриваться папка `c:\accent7.4\forms`.

В файлах расположения допускается указывать относительные пути с использованием специальных символов файловой системы. Точка в этом случае означает текущий каталог, а две точки – родительский каталог.

Директива #include

В модулях *Акцент* реализована дополнительная возможность, позволяющая включать текст одного модуля в другой. Эта возможность не похожа на глобальные модули в других языках. Это означает, что производится именно включение модуля, то есть каждое включение определяет новый экземпляр текста модуля, который компилируется вместе с исходным текстом текущего модуля.

Однако, включение не означает, что текст просто будет подставлен вместо директивы `#include`. Включаемый модуль является именно модулем, со всем присущим ему поведением.

Настоятельно рекомендуется записывать во включаемых модулях только процедуры и функции и не включать другого кода. Это значительно упрощает отладку и сопровождение программ.

Формат директивы `#include` следующий.

```
'#include "filename.ext"
```

Где `#include` – ключевое слово, определяющее саму директиву, а `"filename.ext"` имя включаемого файла. Если полный путь к файлу не указан, то он ищется в соответствии с информацией, которая хранится в файле расположения `ARD`.

По общепринятому соглашению, включаемые файлы имеют расширение `.INC`.

Где используется VBScript

В приложениях семейства *Акцент* язык *VBScript* используется в нескольких элементах системы. Соответственно, каждый такой элемент содержит программный модуль. Далее будут рассмотрены все такие элементы и приведены особенности применения языка *VBScript* в каждом из них.

Модули

Модуль в *Акцент* представляет собой просто программу, которая выполняется независимо от других. Модуль может включать процедуры, и функции. Кроме того, у модуля есть «тело». Это часть программы, которая находится вне каких либо процедур или функций. Тело модуля выполняется непосредственно после его загрузки и запуска.

Существует возможность запустить модуль специальным образом, при котором выполняется только заданная процедура. Для этого используется метод `RunModule` объекта `Application`.

Формы

Форма в среде *Акцент* представляет собой аналог печатного документа, но с возможностью ввода информации. Форма полностью программируемый объект и содержит внутри себя программный модуль на *VBScript*.

Структура программы в модуле формы определяется принципами программирования, построенного на обработке событий. Другими словами, процедуры в модуле выполняются не сами по себе, а только в ответ на происходящие в системе события. Событие может инициировать как пользователь (например, нажав клавишу на клавиатуре) или сама программа (пример такого события – загрузка формы). События присущи как самой форме, так и элементам управления, которые на ней расположены.

Такие процедуры именуются специальным образом:

```
имяОбъекта_имяСобытия
```

Например, при нажатии кнопки с именем `cmdButton` система будет пытаться выполнить процедуру `cmdButton_OnClick`.

Фактически, модуль формы состоит из набора обработчиков событий. Описания формы, а также элементов управления и событий, которые они генерируют, приведены в главе «Формы» на стр. 169.

Диалоги

Диалог в программах Акцент представляет собой стандартное диалоговое окно *Windows*. В отличие от формы, диалог нельзя напечатать. Набор элементов управления, которые могут быть расположены на панели диалога, несколько отличается от элементов управления для форм и включает такие сложные элементы, как дерево, сложный список, таблицу или закладки.

Диалоги, как и формы, включают в себя модуль, содержащий обработчики событий как самого диалога, так и элементов управления, расположенных на панели диалога.

Описания элементов диалогов и событий, которые они генерируют, приведены в главе «Диалоги» на стр. 195.

Электронные таблицы

Помимо того, что электронная таблица, также как и форма или диалог, содержит обработчики событий, имеется еще одно место, где используется *VBScript*. Это формулы в ячейках рабочего листа. Формулы представляют собой обычные выражения языка, в которых допустимо использование адресов ячеек. Формулы всегда начинаются со знака равенства `(=)`. Например, выражение `=A1+B1` означает суммирование значений ячеек `A1` и `B1`

Описание событий, возникающих в процессе работы с электронной таблицей, приведено в главе «Электронные таблицы» на стр. 223.

Модуль Workarea

Один из модулей в программе Акцент имеет специфическое назначение. Этот модуль связан с объектом `Workarea`. Объект `Wokrarea` представляет собой текущую открытую базу данных. Связь модуля с объектом означает, что он содержит обработчики событий, связанных с работой базы данных.

Фактически, этот модуль начинает выполняться при загрузке базы данных и прекращает свое выполнение только при ее закрытии. Однако, это не означает, что модуль `Workarea` является глобальным модулем. Он все равно существует сам по себе. Другими словами, переменные, объявленные в этом модуле, не будут видны в других загруженных модулях, формах, диалогах и электронных таблицах.

Чтобы связать модуль с базой данных, его нужно специальным образом пометить. Это делается в окне проекта текущей базы данных.

Для того, чтобы отметить модуль, связанный с базой данных, нужно открыть окно проекта командой **Сервис | Конструктор**, развернуть в дереве элемент **«Модули»**, выделить нужный модуль и нажать правую кнопку мыши. В появившемся контекстном меню, нужно выбрать команду **«Модуль рабочей области»**. При повторном выборе команды, поставленная ранее отметка снимается. Обратите внимание на то, что для

того, чтобы изменения вступили в силу, требуется перезагрузка базы данных.

Связь модулей, форм, диалогов и таблиц

Модулям приложения *Акцент* необходимо обмениваться между собой информацией. Для обеспечения этой возможности используется следующий подход.

В каждой форме, диалоге или таблице (но не в модуле) имеется специальный объект `Map` (карта).

Объект `Map` можно рассматривать как массив пар *«имя– значение»* (такие объекты часто называют *ассоциативными массивами*). Имя – это просто текстовая строка. Доступ к значениям осуществляется по именам.

После того, как форма или диалог созданы, его карта доступна вызывающей программе через специальное свойство `Map`. Вызывающая программа может записать в него любые значения. Эти значения будут доступны для запущенной программы объекта.

Кроме обычной передачи данных, часто бывает необходимо из одного модуля (например из диалога) инициировать выполнение какого-либо кода в другом модуле (например в вызвавшей этот диалог форме). Для решения этой проблемы в системе предусмотрены *пользовательские события*.

На вызывающей стороне специальный метод `FireUserEvent` программно генерирует событие в другом модуле. На принимающей стороне выполняется обработчик события `OnUser`. Обратите внимание, что хотя метод `FireUserEvent` можно вызывать в любом контексте, обработчик события `OnUser` всегда будет выполняться в текущем контексте.

Метод `FireUserEvent` имеет два аргумента, которые просто передаются в обработчик. Эти аргументы используются для того, чтобы различать, кто послал событие и как его нужно обрабатывать.

Более подробные сведения о пользовательских событиях приведены в электронной справочной системе, поставляемой вместе с программами *Акцент*.

Глава II.

Основы VBScript

В этой главе приведены основные сведения о *VBScript*, описаны используемые типы данных, рассказано о том, что такое переменные и константы. Кроме того, приведены основные концепции объектно-ориентированного программирования, описано, что такое свойства, методы и события и как они используются в программах на *VBScript* в среде *Акцент 7.4*.

Формат программы

Программа на языке *VBScript* состоит из инструкций и комментариев.

Переход на следующую строку в языке является существенным. Перевод строки обычно считается признаком окончания инструкции. Исключением являются составные инструкции, которые могут располагаться в нескольких строках. Для таких инструкций имеется обязательный завершитель. Например, инструкция `For` всегда завершается ключевым словом `Next`.

Для экономии места допускается размещение нескольких инструкций в одной строке. В этом случае они должны разделяться символом двоеточия. Например:

```
A=2 : B=3 : C = A + B
```

Для очень длинных инструкций имеется возможность их размещения в нескольких строках. Признаком продолжения строки является символ подчеркивания, причем, после этого символа, в этой строке не должно быть других символов. Например:

```
MsgBox "Это длинная строка " & _
   "с продолжением"
```

Комментарии

Комментарии используются для включения в текст различных пояснений. Комментарии никак не обрабатываются компилятором и предназначены

для людей, читающих и сопровождающих программу. Хорошим тоном в программировании считается, чтобы исходный код программы содержал исчерпывающее описание применяемых алгоритмов, особенностей применения и т.д.

Комментарий начинается с ключевого слова **Rem** или с одиночной кавычки и продолжается до конца строки. Это означает, что вставить комментарий внутрь инструкции невозможно. Размещение комментариев на нескольких строках не допускается.

Например:

```
Rem -- это комментарий
' -- это еще один комментарий
```

Литералы

Для обозначения константных значений в языке используются литералы. Тип литерала определяется его значением. Возможны следующие литералы.

Тип	Описание
Целое число	Обычное число без десятичной точки, возможно со знаком минус.
Вещественное число	Число с десятичной точкой, возможно со знаком и порядком после букв **E** или **e**.
8-ричное число	Число в 8-ричной системе с префиксом **&O**.
16-ричное число	Число в 16-ричной системе с префиксом **&H**.
Строка	Текстовая строка в двойных кавычках. Чтобы вставить в строку кавычку, ее нужно удвоить.
Дата	Дата в стандартном формате, заключенная в символы **#**.

Имена

Все имена в программе, такие как названия переменных, функций, констант, должны удовлетворять стандартным соглашениям.

Имена могут содержать произвольное количество букв латинского алфавита, цифр и символов подчеркивания. Имена всегда должны начинаться с буквы или с символа подчеркивания. В языке не учитывается регистр символов, поэтому имена **totalcash**, **TotalCash** и **TOTALCASH** считаются одинаковыми.

Имена не могут совпадать с зарезервированными словами языка. Полный список зарезервированных слов приведен в приложении **А**.

Типы данных

Язык *VBScript* относится к безтиповым языкам. Это означает, что все переменные и константы представлены только одним типом данных – `Variant`**.** Данные типа `Variant` могут содержать любую информацию. Это могут быть числа, строки, даты, денежные величины, булевские значения, объекты и даже многомерные массивы. Каждый из типов данных, которые хранятся в переменной типа `Variant`**,** называется подтипом.

В следующей таблице приведен список подтипов данных, поддерживаемых в языке *VBScript*.

Тип	Константа	Описание
`Empty`	`vbEmpty`	Пустое значение
`Null`	`vbNull`	Значение `Null` (не нуль!)
`Boolean`	`vbBool`	Булево значение (`True` или `False`)
`Integer`	`vbInteger`	Целое число со знаком в диапазоне -32768…+32767
`Long`	`vbLong`	Длинное целое со знаком в диапазоне -2^{31}…$+2^{31}-1$
`Single`	`vbSingle`	Число с плавающей точкой одинарной точности в диапазоне примерно $\pm1.0^{-38}$…1.0^{+38}
`Double`	`vbDouble`	Число с плавающей точкой двойной точности в диапазоне примерно $\pm1.0^{-300}$…1.0^{+300}
`Currency`	`vbCurrency`	Денежные значения в диапазоне примерно ± 900 000 000 000 000 с четырьмя знаками после точки.
`Date`	`vbDate`	Дата и/или время между 1 января 0100г. до 31 декабря 9999 г.
`String`	`vbString`	Текстовая строка длиной до 2 Гбайт.
`Object`	`vbObject`	Ссылка на объект
`Error`	`vbError`	Код ошибки (системное значение)

В языке поддерживается автоматическое преобразование от одного подтипа к другому везде, где это имеет смысл.

Например, если текущее значение переменной имеет тип `Long`, а в контексте требуется тип `String`, то будет выполнено автоматическое преобразование числа в строку.

Интересно то, что возможно преобразование объектного типа, например, в текстовую строку. В этом случае используется понятие «свойство по умолчанию (`Default`)». Например, если по контексту программы требуется строковое значение, а имеется объект, то *VBScript* пытается найти у этого объекта свойство «по умолчанию» и, если оно есть – получает его значение. Если результат снова объект, процесс повторяется до тех пор, пока не получится значение, которое можно преобразовать в строку.

У программиста всегда имеется возможность определить, к какому подтипу относится текущее значение переменной. Для этого используется встроенная функция `VarType()`.

Примечание. Не путайте значения переменных и константы, которые описывают эти значения. Например, если переменная равна `Null`, то у нее нет никакого значения, а функция `VarType()` вернет единицу. Эта единица (константа `vbNull`) и означает, что переменная равна `Null`.

Переменные и константы

Переменные

Переменная – это просто место в памяти компьютера, где могут храниться данные. Переменная имеет имя, с помощью которого осуществляется доступ к значению переменной. Объявлять переменные не обязательно, хотя настоятельно рекомендуется. Если переменная не объявлена явно, то ее объявлением считается первое упоминание о ней в левой части инструкции присваивания. Для явного объявления переменной используются инструкция `Dim`. Вы можете прямо указать, что требуется явно объявлять все переменные. Для этого поместите в начало модуля инструкцию `Option Explicit`.

Обратите внимание, что явное объявление переменных помогает избежать появления трудноуловимых ошибок из-за неверного написания имен, поэтому использовать `Option Explicit` желательно всегда.

Константы

Константа это значение, которое не изменяется в ходе выполнения программы. Константы объявляются с использованием ключевого слова `Const`. Одновременно с объявлением, константе присваивается значение. Попытка изменить это значение в программе приведет к ошибке выполнения. Тип данных константы определяется ее значением. Например:

```
Const strText = "Это строковая константа"
Const nMin = 2 ' минимальное значение
```

В системе имеется большое количество встроенных констант. Их не нужно объявлять, а можно просто использовать. Константы, встроенные в *VBScript*, имеют префикс `vb` и обычно записываются «лесенкой», например, `vbUseSystem`. В программах *Акцент* также имеются собственные константы. Они начинаются с префикса `ac`, например, `acBrdTop`.

Более подробную информацию о встроенных константах можно получить в электронной справочной системе.

Объекты и коллекции

Объекты

Объекты – основа любого объектно-ориентированного приложения. Все сущности программы, такие как счет, корреспондент, бухгалтерская операция или проводка представлены объектами. Если говорить коротко, объект это нечто цельное, чем можно управлять и что можно программировать. Каждый из объектов предназначен для хранения внутри определенной информации и для выполнения каких-либо действий. Разработка объектно-ориентированных программ заключается в объединении нужных объектов средствами языка программирования.

Объекты могут содержать внутри себя другие объекты, создавать другие объекты по запросу. Вся совокупность объектов внутри программы и взаимосвязь их между собой называется *объектной моделью* приложения. Понимание объектной модели – ключ к разработке хороших программ, которые легко разрабатывать и удобно сопровождать.

Классы и экземпляры

Любой объект в системе принадлежит к одному из *классов*. Класс или другими словами *тип* объекта – это общая характеристика всей совокупности объектов. Конкретное воплощение класса, описывающее характе-

ристики конкретного объекта, называется *экземпляром*. Например, в системе имеется класс объектов `Agent`, описывающий корреспондентов. Все корреспонденты в системе имеют одинаковый набор характеристик. Конкретный объект, описывающий, к примеру, фирму «Импакт», является экземпляром класса `Agent`.

Для работы с конкретным экземпляром объекта нужно иметь *ссылку* на него. В языке *VBScript* ссылка часто называется *объектной переменной*. Ссылку на объект можно получить различными способами, но чаще всего, ее возвращает какое-либо свойство или метод другого объекта, более высокого уровня. Отсюда автоматически следует, что объектная модель приложения будет иерархической, – другими словами существует объект верхнего уровня, внутри которого содержатся все остальные объекты. В программах *Акцент* (как и во многих других) объект верхнего уровня представляет собой само приложение и так и называется `Application`.

Примечание. Обратите внимание, что при работе со ссылками на объекты используется специальная инструкция `Set`. Если Вы забудете указать `Set`, то программа или не будет работать вообще, или будет делать совсем не то, что требовалось. В частности, если попробовать присвоить обычной переменной значение ссылки (не используя `Set`), то система попытается найти у объекта свойство «по умолчанию» (`Default`) и использовать его. Более подробно работа с инструкцией `Set` описана в главе «Инструкции VBScript».

Свойства методы и события

Все объекты можно описать. Каждому из объектов соответствует набор атрибутов, определяющих различные описательные параметры. Такие параметры называются *свойствами*.

Свойства – это атрибуты объекта, определяющие его характеристики, такие, например, как имя, размер, цвет. Некоторые свойства можно только читать, но не записывать. Для обращения к свойству объекта нужно записать объектную переменную, а затем, после точки, имя свойства. Например:

```
' чтение свойства
clr = object.Color

' установка свойства
object.Color = vbRed ' установить красный цвет
```

Методы – это действия, которые можно выполнить над объектом или может выполнить сам объект. В результате выполнения метода может измениться состояние объекта (и соответственно, могут измениться значения некоторых свойств), могут быть созданы новые объекты и так далее. Если свойства обычно будут описываться прилагательными, то методы – это глаголы. Синтаксис обращения к методам такой же, как и к свойствам.

Многие объекты системы, такие, например, как формы или диалоги специально предназначены для взаимодействия с пользователем. Для того, чтобы они могли откликаться на действия пользователя, они поддерживают *события*.

Событие – это то, что происходит в системе. Это могут быть действия пользователя: нажатие кнопки, щелчок мышью, или действия самой программы, например, сохранение объекта в базе данных. Для того, чтобы обработать событие, нужно написать специальную подпрограмму, называемую *обработчиком события*. При возникновении события система ищет соответствующий обработчик и выполняет его. Имя обработчика события определяется следующим образом:

```
имяОбъекта_имяСобытия
```

Например, при нажатии кнопки с именем `cmdButton` система будет пытаться выполнить процедуру `cmdButton_OnClick`.

Примечание. Обратите внимание, что имя объекта и текст, который на нем отображается, это не одно и то же. Имя объекта никогда не видно пользователю и служит только для обозначения объекта в программе. Например, на кнопке с именем `cmdButton` вполне может быть написано что-то вроде «Выполнить».

Многие события имеют дополнительные параметры. Для того чтобы, система могла найти нужный обработчик, его аргументы должны соответствовать параметрам события. Интерфейс работы с модулями в программах *Акцент* поддерживает автоматическую вставку обработчиков событий со всеми необходимыми аргументами.

Коллекции

Если не вдаваться в подробности, то коллекция – это просто группа объектов, принадлежащих одному классу. Хотя коллекция объединяет в себе несколько объектов, она сама также является объектом. Существует пра-

вило, согласно которому объект обычно называется существительным в единственном числе, а семейство таких объектов существительным во множественном числе. Например, объект `Agent` представляет собой корреспондента, а коллекция `Agents` – коллекцию корреспондентов. В самом упрощенном варианте можно рассматривать коллекцию просто как массив объектов.

Поскольку коллекции сами являются объектами, то они поддерживают набор свойств и методов. Некоторые из них являются общими практически для всех коллекций. Это свойство `Count`, возвращающее количество объектов в коллекции и `Item`, позволяющее получить ссылку на содержащийся в коллекции объект. Очевидно, что свойство `Item` обязательно имеет аргумент – номер объекта в коллекции. Нумерация объектов в коллекции, как правило, начинается с единицы.

Глава III.

Инструкции VBScript

В этой главе описаны инструкции языка *VBScript.* Для каждой инструкции дается описание синтаксиса, особенностей использования и приводятся примеры.

Программа на языке *VBScript* состоит из инструкций, операций и идентификаторов (имен). Инструкции – это команды языка, предписывающие системе выполнить какое-либо действие. Инструкции составляются из ключевых слов, изменять написание которых нельзя. Примеры ключевых слов:

```
If  Then  Else  End  Sub  Function
```

Полный список ключевых слов языка *VBScript* приведен в **приложении А**.

Операции языка представляют собой обычные арифметические операции, такие как сложение, вычитание, умножение, и деление. Помимо этих, существуют логические операции для работы с условными выражениями, а также некоторые другие.

Идентификаторы языка представляют собой имена переменных, констант, подпрограмм и функций. Идентификаторы могут содержать произвольное количество букв латинского алфавита, цифр и символов подчеркивания. Идентификаторы всегда должны начинаться с буквы или с символа подчеркивания. В языке не учитывается регистр символов, поэтому имена **totalcash**, **TotalCash** и **TOTALCASH** считаются одинаковыми.

Особым образом именуются процедуры обработки событий. Имя процедуры всегда состоит из имени объекта, символа подчеркивания и имени события. Например, процедура обработки события `OnClick`, которое будет возникать при нажатии на кнопку с именем `Button1`, будет называться `Button1_OnClick`.

Объявление переменных

Объявлять переменные в языке *VBScript* необязательно. Переменная считается автоматически объявленной при первом ее появлении в левой части инструкции присваивания. Однако такой подход может привести к трудноуловимым ошибкам. Если, например, Вы случайно неверно напишите одну букву в имени переменной, то VBScript объявит новую переменную, и понять, что происходит, будет очень трудно.

Для того, чтобы избежать этой проблемы, в *VBScript* была добавлена специальная инструкция `Option Explicit`, которая помещается в начале модуля и говорит о том, что все переменные в модуле должны быть объявлены явно, с помощью инструкции `Dim`. Поскольку такая практика общепринята, в программах семейства *Акцент* инструкция `Option Explicit` всегда автоматически включается в начало каждого модуля.

Инструкция Dim

Инструкция `Dim` объявляет переменные и, возможно, выделяет для них память. Если `Dim` располагается внутри инструкции `Class`, то это описание члена данных класса. Никакая память в этом случае не выделяется.

Синтаксис инструкции следующий.

```
Dim Name[([Size])][,Name[([Size])]]...
```

Где `Name` представляет собой имя объявляемой переменной, а Size размерности переменных типа массива. Допускается задавать до 60 измерений массива. В качестве размерности указываются верхние границы измерений через запятую. Нижние границы всегда равны 0.

Если объявить массив с пустыми круглыми скобками, то это значит, что объявлен динамический массив, размерность которого будет задана позже инструкцией `ReDim`.

В одной инструкции `Dim` можно объявлять несколько переменных (через запятую).

После объявления переменные инициализируются нулем (для чисел) и пустой строкой (для строк). Для объектных переменных значение будет равно `Nothing`.

Если указаны скобки и измерения массива, то его размеры изменить невозможно.

Массивы

Массивы в языке *VBScript* могут быть нескольких типов. Во-первых, это обычные массивы. Их размер определяется в момент объявления и не может изменяться в процессе выполнения программы. Массивы могут быть многомерными и иметь до 60 измерений. Например:

```
Dim a(5)   ' массив из 5 элементов
Dim b(5,5) ' матрица 5x5 элементов
```

Индексация в обычных массивах всегда начинается с нуля.

Часто размер массива заранее неизвестен. В этом случае полезен так называемый *открытый или динамический массив*. Такой массив объявляется инструкцией `Dim` с пустыми круглыми скобками.

```
Dim array() ' динамический массив
```

Его размер устанавливается (или переустанавливается) в процессе выполнения программы с помощью инструкции `ReDim`. Изменить размеры массива можно так, чтобы хранимые в нем данные сохранились. Для этого служит специальное ключевое слово `Preserve`. Само собой разумеется, что, если нужно сохранить данные в многомерном динамическом массиве, то можно менять значение только последнего измерения массива.

Помимо этого, возможно создание обычных переменных типа `Variant`, которые содержат массивы. Такие массивы либо возвращаются различными объектами, либо создаются с помощью встроенной функции `Array()`.

Примечание. Обратите внимание, что массив, созданный функцией `Array`, будет индексироваться с нуля. Для массивов, которые возвращают объекты, это совершенно не обязательно. Они могут индексироваться с любого удобного значения, не обязательно с нуля или с единицы. Для определения размеров таких массивов используются функции `LBound` и `RBound`

Отметим, что переменная типа `Variant`, содержащая массив, это не тоже самое, что массив переменных типа `Variant`, хотя синтаксис обращения к элементу массива будет выглядеть одинаково.

Можно создать переменную типа `Variant`, содержащую массив, каждый из элементов которого, в свою очередь, является массивом и т.д. Это будет не многомерный массив, а будет массив массивов. Доступ к элемен-

там второго уровня в этом случае возможен только через промежуточную переменную. Конечно, лучше избегать таких сложных случаев, хотя иногда это может быть удобно.

Инструкция Erase

Инструкция `Erase` очищает (заново инициализирует) массив. Синтаксис инструкции следующий:

```
Erase arg
```

Где `arg` является именем переменной массива.

Для массивов фиксированного размера инструкция `Erase` не изменяет количество элементов массива, а просто очищает его. Очистка заключается в том, что всем элементам массива присваиваются пустые значения.

Если массив динамический, то применение инструкции `Erase` приведет к тому, что содержимое массива будет уничтожено и его размер станет равным нулю. Для повторной инициализации массива нужно будет снова использовать инструкцию `ReDim`. То же самое происходит и с переменными типа Variant, которые содержат массивы. После применения инструкции `Erase` к такому массиву, он станет пустым (не будет содержать ни одного элемента).

Инструкция ReDim

Инструкция изменяет размеры существующего динамического массива. Динамический массив создается инструкцией `Dim` с пустыми круглыми скобками.

Синтаксис инструкции `ReDim`.

```
ReDim [Preserve] Name([Size]),[Name([Size])]...
```

Где `Name` представляет собой имя массива, а `Size` – новая размерность массива.

Необязательное ключевое слово `Preserve` говорит о том, что содержимое массива по возможности нужно сохранить.

Пример

```
Dim x()     ' пустой массив
ReDim x(5) ' теперь в массиве 5 элементов
x(4) = 2
' Увеличим размер до 10, сохранив содержимое
```

```
ReDim Preserve x(10)
MsgBox x(4) ' Выведет 2
```

Присваивание

Инструкция присваивания по праву считается основной в любом языке программирования. Ее синтаксис чрезвычайно прост:

```
Var = Expression
```

Где `Var` представляет собой имя переменной, элемент массива или свойство, доступное для записи. `Experssion` представляет собой выражение, результат вычисления которого будет присвоен переменной `Var`.

Выражение, которое находится в правой части после знака равенства, представляет собой обычное арифметическое или логическое выражение. В него могут входить переменные, элементы массивов, свойства объектов (которые можно читать), а также вызовы функций. Для управления порядком вычисления выражения можно использовать круглые скобки.

Инструкция Set

Присваивает переменной или значению свойства ссылку на объект или создает новый объект заданного класса. Свойства, значения которых присваиваются инструкцией `Set`, также должны представлять собой объекты.

Синтаксис инструкции `Set` следующий.

```
Set Var = Expression | Nothing | New ClassName
```

Где `Var` представляет собой имя объектной переменной или свойство, представляющее собой объект. **`Experssion`** – выражение, результатом которого является объект. Это может быть, например, свойство объекта или вызов функции. Очевидно, что арифметические операции с объектами бессмысленны.

Если инструкция Set используется для создания экземпляра класса, то **`ClassName`** – это имя класса для создаваемого объекта. Класс должен быть описан в инструкции `Class`.

`Nothing` – специальное ключевое слово, определяющее, что переменная не ссылается ни на какой объект.

Примечание. Все ссылки на объекты обязательно должны присваиваться инструкцией `Set`. Обычная инструкция присваивания приведет к ошибке времени выполнения. Для того, чтобы определить, ссылается ли переменная хоть на что-нибудь, используется операция `Is`.

Операции и выражения

Арифметические операции

В языке *VBScript* имеется полный набор арифметических операций. Они используются для выполнения действий над числами. Определены следующие арифметические операции:

- **+** – сложение
- **–** – вычитание
- ***** – умножение
- **/** – деление
- **** – деление нацело
- **Mod** – остаток от целого деления
- **^** – возведение в степень

Обратите внимание, что если один из аргументов операции равен `Null`, то результат будет равен `Null`. Значение `Empty` интерпретируется как нуль.

Примечание. Операция сложения **(+)** может использоваться для конкатенации (сцепления) строк. Однако ее поведение может отличаться от того, что Вы ожидаете. Поскольку это все-таки сложение, то система пытается преобразовать аргументы к числам. Однако, если оба аргумента строки, то просто выполняется их конкатенация. Поэтому 2 + "2" будет равно 4, а "2" + "2" будет равно "22". Чтобы избежать этих проблем, для строк рекомендуется всегда использовать операцию конкатенации (&).

Операции сравнения

Операции сравнения используются там, где нужно проверить какое-либо условие. Чаще всего это условные инструкции и циклы. В *VBScript* определены следующие операции сравнения:

- **>** – больше
- **<** – меньше
- **>=** – больше или равно

<= – меньше или равно
= – равно
<> – не равно

Обратите внимание, что если один из аргументов операции равен `Null`, то результат будет равен `Null`. Значение `Empty` интерпретируется как 0.

Например, в следующем фрагменте, проверяется, что переменная `val` больше нуля:

```
If val > 0 Then
   ' что-то делаем
End If
```

Примечание. Знак равенства **(=)** используется в языке в двух смыслах: как знак операции присваивания и как операция «равно». Обычно это не приводит к проблемам, так как конкретный смысл знака равенства легко определяется по контексту.

Логические операции

Логические операции используются для объединения операций сравнения в сложные выражения. В языке имеются следующие логические операции:

And – логическое «И»
Or – логическое «ИЛИ»
Xor – исключающее «ИЛИ»
Imp – импликация
Eqv – эквивалентность
Not – логическое «НЕ»

Логические операции выполняются в соответствии со стандартным правилами булевой алгебры.

Чаще всего, логические операции используются в условных инструкциях и в циклах по условию.

Операция конкатенации

Операция конкатенации используется для «сцепления» строк. Операция обозначается символом **&**. Результатом операции тоже будет текстовая строка. Если операнды не являются строками, то система попытается пре-

образовать их в строки. Чтобы избежать неоднозначности при этих попытках, используйте встроенную функцию `CStr`.

Операция Is

Операция `Is` используется для проверки эквивалентности ссылок на объекты. Операция возвращает `True`, если две ссылки указывают на один и тот же объект. Операция не выполняет никаких действий с объектами, более того, объекты даже могут не существовать. Особым случаем использования `Is` является проверка на существование объекта. Для этого используется ключевое слово `Nothing`, означающее «ничего». Например:

```
If obj Is Nothing Then
  MsgBox "Объекта нет"
Else
  MsgBox "Имя объекта = " & obj.Name
End If
```

Примечание. Обратите внимание, что ключевое слово `Nothing` можно использовать только в операции `Is` и в инструкции `Set`.

Порядок выполнения операций

Когда в выражении встречается несколько операций, то порядок их выполнения строго определен правилами языка. Стандартный порядок выполнения операций приведен в следующей таблице.

Арифметические		Сравнение		Логические	
Операция	Знак	Операция	Знак	Операция	Знак
Степень	^	Равно	=	НЕ	Not
Минус	-	Не равно	<>	И	And
Умножение	*	Меньше	<	ИЛИ	Or
Деление	/	Больше	>	Искл. ИЛИ	Xor
Деление нацело	\	Меньше, равно	<=	Эквивалент.	Eqv
Модуль	Mod	Больше, равно	>=	Импликация	Imp
Сложение	+	Операция Is	Is		
Вычитание	-				
Конкатенация	&				

Если в выражении встречаются арифметические и логические операции, а также операции сравнения, то сначала всегда выполняются арифметические операции, затем операции сравнения. Логические операции выполняются в последнюю очередь.

Операция конкатенации строк (&) не является арифметической, но выполняется до любых операций сравнения.

Управляющие инструкции

Язык *VBScript* поддерживает полный набор управляющих инструкций.

Инструкция If...Then...Else

Для принятия решений в программе используется условная инструкция `If Then Else`.

```
If condition Then
  yes_statements
End If
```

или полный вариант:

```
If condition Then
  yes_statements
Else
  no_statements
[ElseIf
  elseif_statements]
End If
```

Вначале проверяется условие `condition`. Если оно равно `True`, то выполняется группа инструкций `yes_statements`. Если условие равно `False`, то будет выполнена группа инструкций `no_statements`.

Необязательный блок `ElseIf` позволяет проверять в одной инструкции сразу несколько условий.

Возможна запись инструкции в одну строку. В этом случае завершающий `End If` не нужен.

```
If condition Then statement [Else statement]
```

Пример

```
If i = 1 Then
```

```
   MsgBox "i = 1"
ElseIf i = 2 Then
   MsgBox "i = 2"
Else
   MsgBox "i <> 1 and I<>2"
End If
```

Инструкция Select…Case

Если нужно выбрать путь из нескольких альтернатив, то вместо использования сложной инструкции `If…Then…Else`, можно воспользоваться инструкцией `Select…Case`.

Синтаксис инструкции следующий.

```
Select Case test
   Case value1
      value1_statements
   [Case valueN
      valueN_statements]
[Case Else
   else_statements]
End Select
```

Где `test` представляет собой числовое или строковое выражение, значение которого будет использоваться для ветвления. `value1…valueN` представляет собой список значений (через запятую). Если значение `test` совпадает с одним из значений из `valueN`, то выполняется соответствующая группа инструкций `valueN_statements`.

Необязательный блок `Case Else` будет выполнен в том случае, если совпадений в других частях инструкции не найдено.

Пример

```
Select Case i
   Case 1
      MsgBox "i = 1"
   Case 2
      MsgBox "i = 2"
   Case Else
      MsgBox "i <> 1 and I<>2"
End Select
```

Инструкция For…Next

Цикл `For…Next` предназначен для выполнения блока кода заданное число раз. Такие циклы применяются тогда, когда заранее известно, сколько раз нужно повторить блок кода. Например, если нужно перебрать элементы массива, то число повторений будет равно измерению массива.

Цикл `For…Next` имеет следующий синтаксис:

```
For counter=start To end [Step step]
   statements
   [Exit For]
   [statements]
Next
```

Где `counter` (счетчик цикла) является именем переменной, в которой будет храниться текущее количество повторений цикла. Значения `start` (начало) и `end` (конец) определяют начальное и конечное значение счетчика. Необязательный элемент цикла – `step` (шаг). Он определяет шаг приращения счетчика при каждом повторении. Инструкции `statements` являются *телом* цикла, то есть именно они будут многократно выполняться.

Когда начинается выполнение цикла, переменная `counter` получает значение `start`, затем выполняются инструкции цикла вплоть до `Next`. Далее `counter` увеличивается на величину шага `step` и сравнивается со значением `end`. Если `counter` больше `end`, цикл прерывается и управление передается на инструкцию, непосредственно следующую за `Next`. Так продолжается до тех пор, пока значение `counter` не станет больше `end`.

Цикл можно «досрочно» прервать в любом месте с помощью инструкции `Exit For`. Когда в процессе выполнения цикла дело доходит до `Exit For`, управление немедленно передается на инструкцию, следующую за `Next`. Счетчик цикла в этом случае сохраняет свое значение. Это обстоятельство можно использовать для того, чтобы узнать, завершился ли цикл обычным образом, или досрочно. Чаще всего `Exit For` используется в циклах поиска, когда искомое значение найдено.

Необязательный элемент цикла `Step`. Он задает шаг приращения цикла. Обратите внимание, что `step` может быть отрицательным. Тогда при каждом повторении цикла, счетчик не увеличивается, а уменьшается. Если `step` не указан, то он считается равным единице.

Переменная, которая используется в качестве счетчика – это обычная переменная, и ее вполне можно изменить внутри цикла. Так делать можно, но не нужно, это может привести к непредсказуемым результатам. В этом случае логика программы становится крайне запутанной. Например, следующий фрагмент кода будет выполняться вечно:

```
For i = 1 To 10
   i = i - 1
Next
```

Приведем пример программы, использующей цикл `For…Next`. В данном примере осуществляется поиск заданного значения в массиве.

```
' функция ищет what в array и возвращает
' найденный индекс. Если не нашла, возвращает -1

Function Find(array,what)
   Find = -1 ' считаем, что не нашли
   For i=LBound(array) To UBound(array)
      If array(i) = what Then
         Find = i ' нашли, вернем i
         Exit Function
      End If
   Next
End Sub
```

Инструкция Do…Loop

Цикл `Do…Loop` используется для выполнения некоторого блока кода до тех пор, пока выполняется или не выполняется какое-либо условие. Если цикл `For…Next` выполняется строго определенное количество раз, то цикл `Do…Loop` будет выполнять блок кода неопределенное количество раз, или вообще не выполнять. Все это зависит от логического условия. Цикл `Do…Loop` является наиболее универсальным в языке *VBScript*. Существуют четыре варианта этого цикла: условие `True` в начале, условие `True` в конце, условие `False` в начале и условие `False` в конце. Если условие цикла проверяется в начале, такой цикл называется циклом с *предусловием*, а если в конце, то с *постусловием*.

Синтаксис всех четырех вариантов приведен ниже:

```
' условие True в начале
Do While condition
   statements
```

```
   [Exit Do]
   [statements]
Loop
'
' условие True в конце
Do
   statements
   [Exit Do]
   [statements]
Loop While condition
'
' условие False в начале
Do Until condition
   statements
   [Exit Do]
   [statements]
Loop
'
' условие False в конце
Do
   statements
   [Exit Do]
   [statements]
Loop Until condition
```

Ключевые слова `While` и `Until` определяют логику проверки условия. `While` указывает, что цикл будет выполняться до тех пор, пока условие остается истинным, а `Until` – до тех пор, пока условие не станет истинным. Соотношение между `While` и `Until` можно выразить так:

```
While condition = Until Not Condition
```

Как и для цикла `For…Next` инструкция `Exit Do` предназначена для досрочного безусловного выхода из цикла. Кроме того, совместно с инструкцией `If…Then` можно организовать проверку условия не только в начале или в конце цикла, но и в любом месте в середине цикла.

Запись условия в начале или в конце цикла определяет, когда это условие будет проверяться. Если условие проверяется в начале, то цикл может не выполниться ни разу (если условие исходно является ложным). В противоположность этому, если условие записано в конце, то тело цикла всегда выполнится хотя бы один раз.

В приведенном ниже примере цикл Do…Loop используется для перебора всех записей в наборе данных, полученных при выполнении SQL-запроса.

```
Dim db
Dim rs
Dim sql
sql = "select * from TEST_TABLE"
Set db = Workarea.DaoDatabase
Set rs = db.OpenRecordset(sql)
' перебор записей, пока
' не достигнут конец файла
Do Until rs.EOF
   rs.MoveNext
Loop
```

Обратите внимание, что если набор записей пуст, то цикл не выполнится ни разу, поскольку условие проверяется в начале цикла, и, если оно ложно, управление передается следующей за Loop инструкции.

Инструкция For…Each

Цикл For…Each не имеет прямых аналогов в других языках программирования. Он предназначен для перебора массивов и коллекций. Тело цикла будет выполнено по одному разу для каждого элемента массива или члена коллекции. При этом счетчиком цикла будет специальная переменная, которая последовательно будет принимать значения элементов массива или коллекции.

Цикл For…Each имеет следующий синтаксис:

```
For elem In collection
   statements
   [Exit For]
   statements
Next
```

Где elem является именем переменной, которой последовательно будут присваиваться значения элементов массива или коллекции collection.

В следующем примере показано использование цикла For…Each для того, чтобы найти сумму массива чисел.

```
Function Sum(array)
   Dim elem ' один элемент
   Dim val  ' среднее значение
```

```
    val = 0  ' начальное значение равно нулю
    For Each elem In array
       val = val + elem
    Next
    Sum = val
End Function
```

Примечание. Обратите внимание, что в данном случае использование временной переменной `val` обязательно. В теле цикла нельзя использовать имя функции в качестве переменной. Если написать

```
Sum = Sum + val,
```

то это будет означать рекурсивный вызов этой же функции `Sum`. Скорее всего, хотелось получить что-то другое. Рекомендуется принять за правило никогда не использовать возвращаемое значение функции в выражениях.

Инструкция While...Wend

Цикл `While…Wend` считается устаревшим и сохранен для совместимости с предыдущими версиями языка *Basic*. Он сравнительно удобен в простых случаях цикла с предусловием, однако имеет один существенный недостаток: из него нельзя выйти досрочно, то есть инструкции `Exit While` не существует.

Синтаксис цикла `While…Wend` очень прост:

```
While condition
   statements
Wend
```

Такой цикл работает полностью аналогично циклу `Do…Loop` с проверкой условия `True` в начале цикла, то есть:

```
Do While condition
   statements
Loop
```

В приведенном ниже примере делается то же самое, что и в примере для цикла `Do…Loop`.

```
Dim rs
Set rs = Workarea.DaoDatabase.OpenRecordset( _
  "select * from TEST_TABLE")
' перебор записей до конца файла
```

```
While Not rs.EOF
   rs.MoveNext
Wend
```

Функции и процедуры

Для достаточно больших программ неизбежно их разбиение на отдельные части, имеющие самостоятельное значение. Такие части можно затем использовать повторно. За счет этого улучшается структура программы и повышается ее надежность. Грамотная структуризация программы облегчает понимание и упрощает внесение изменений, что снижает последующие затраты на сопровождение.

В языке *VBScript* реализованы два классических способа структуризации программ – процедуры (называемые так же подпрограммами) и функции.

Процедура – это самостоятельно используемая простая программа, которую можно вызвать из другой процедуры. Функция, в отличие от процедуры, возвращает значение, поэтому ее можно использовать в выражениях, так же как, и встроенные функции языка.

В процедуры и функции можно передавать аргументы. Передавать их можно как *по ссылке*, так и *по значению*. Передача по ссылке (определяется ключевым словом `ByRef`) означает, что в процедуру передается не значение переменной, а «сама» переменная. Если поменять значение внутри процедуры, то оно поменяется и в вызывающей программе.

Если аргумент передается по значению (определяется ключевым словом `ByVal`), то изменение значения внутри ни на что снаружи не влияет. Обратите внимание, что объектные переменные всегда передаются по ссылке, даже если указан модификатор `ByVal`.

Инструкция Sub

Инструкция `Sub` описывает процедуру.

Инструкция `Sub` имеет следующий синтаксис:

```
Sub SubName[(arglist)]
   statements
   [Exit Sub]
   statements
End Sub
```

Где SubName является именем процедуры, с помощью которого она будет вызываться, а arglist представляет собой список аргументов, разделенных запятыми. Если аргументов нет, то скобки можно опустить.

Вложенные процедуры недопустимы. Однако процедуры могут вызывать сами себя. Такие процедуры называются *рекурсивными*.

Инструкция Exit Sub приводит к немедленному завершению выполнения процедуры. Программа будет продолжена с инструкции, непосредственно следующей за вызовом процедуры. Внутри процедуры можно использовать любое количество инструкций Exit Sub.

Инструкция Function

Инструкция Function описывает имя, аргументы и тело пользовательской функции.

Инструкция Function имеет следующий синтаксис:

```
Function SubName[(arglist)]
   statements
   [SubName = expression]
   statements
   [Exit Function]
   statements
   [SubName = expression]
   statements
End Function
```

Инструкция Exit Function приводит к немедленному завершению выполнения функции. Внутри функции можно использовать любое количество инструкций Exit Function. Не забывайте присвоить возвращаемое значение перед выходом из функции.

Инструкция Call

Инструкция Call предназначена для вызова процедур. Она считается устаревшей, хотя ей можно найти очень удобное применение. Дело в том, что, если при вызове процедуры перед ее именем поставить Call, то аргументы процедуры нужно будет заключить в круглые скобки, как при вызове функции.

Например,

```
MsgBox "Это сообщение"  ' правильно
```

```
MsgBox("Еще сообщение") ' неправильно
Call MsgBox("Ура!")     ' снова правильно
```

Вспомогательные инструкции

Инструкция Execute

Инструкция `Execute` рассматривает текстовую строку как часть программы на языке *VBScript* и выполняет ее. Чтобы выполнить несколько инструкций, можно разделить их двоеточием.

Инструкцию можно использовать для динамической генерации кода в процессе выполнения программы.

Например,

```
Dim x
Execute "x = 2*2"
MsgBox(x) ' выведет значение 4
```

Имеется вариант инструкции `ExecuteGlobal`, который выполняет код в глобальном контексте. Это означает, что даже если инструкция вызвана из процедуры, она будет выполняться так, как будто бы она была вызвана вне этой процедуры.

Глава IV.

Встроенные функции

В этой главе будут описаны все встроенные функции языка *VBScript* и программ семейства *Акцент*. Описания сгруппированы по семействам функций, а внутри семейства по алфавиту.

Преобразование данных

Хотя все переменные в языке имеют один и тот же тип (`Variant`) преобразование типов данных все равно бывает необходимо. Всегда, когда нужно быть уверенным в том, что переменная относится к определенному подтипу, используются функции преобразования. В *VBScript* имеется обширный набор встроенных функций для преобразования одного подтипа данных в другой. Перечень встроенных функций преобразования приведен в следующей таблице (в алфавитном порядке).

Функция	Описание
`CBool`	Преобразовать в булев тип (`Boolean`)
`CByte`	Преобразовать в байт (целое 0..255)
`CCur`	Преобразовать в денежный тип (`Currency`)
`CDate`	Преобразовать в дату (`Date`)
`CDbl,CSng`	Преобразовать в число с плавающей точкой (`Double` или `Single`)
`CInt,CLng`	Преобразовать в целое число (`Integer` или `Long`)
`CStr`	Преобразовать в строку (`String`). Функция описана в разделе, посвященном строковым функциям (стр. 62).
`Hex, Oct`	Возвращает шестнадцатеричное или восьмеричное представление числа. Функции описаны в разделе, посвященном строковым функциям (стр. 65).

Далее приведено краткое описание каждой функции с примерами применения.

Функция CBool

Функция преобразовывает выражение в булево значение `True` или `False`. Если аргумент равен нулю, возвращается `False`, если нет – `True`. Если аргумент нельзя интерпретировать как число, происходит ошибка выполнения

Синтаксис

```
CBool(value)
```

Аргументы

`value` – выражение для преобразования.

Пример

```
Dim res
res = CBool(1) ' вернет True
res = CBool(0) ' вернет False
```

Функция CByte

Функция преобразовывает выражение в байт (`Byte`) (число в диапазоне 0..255). Если аргумент нельзя интерпретировать как число или оно выходит за пределы заданного диапазона, происходит ошибка выполнения.

Синтаксис

```
CByte(value)
```

Аргументы

`value` – выражение для преобразования.

Пример

```
Dim res
res = CByte("122") ' вернет 122
res = CByte(256)   ' вызовет ошибку
```

Функция CCur

Функция преобразовывает выражение в денежный тип (`Currency`). Если аргумент нельзя интерпретировать как число, происходит ошибка выполнения. Если дробная часть имеет длину больше четырех знаков, то значение округляется. Обратите внимание, что при преобразовании текстовых строк, система будет учитывать региональные установки *Windows*. Это касается разделителя целой и дробной частей числа, разделителя тысяч и способа представления отрицательных чисел.

Синтаксис

CCur(value)

Аргументы

value – выражение для преобразования.

Пример

```
Dim res
res = CCur(12.23229) ' вернет 12.2323
```

Примечание. *VBScript* использует «банкирский» способ округления, когда 0.5 округляется до ближайшего ЧЕТНОГО целого. В *Акцент* встроена функция Round2, которая работает традиционным способом.

Функция CDate

Функция преобразовывает выражение в дату (Date). Если аргумент нельзя интерпретировать как дату или время, происходит ошибка выполнения программы. Чтобы избежать ошибок выполнения, можно воспользоваться функцией IsDate(), которая проверяет, можно ли интерпретировать аргумент как дату.

Дата может быть представлена в произвольном формате. Обратите внимание, что при преобразовании текстовых строк система будет учитывать региональные установки *Windows*.

Если в дате не указан год, то берется текущий год. При преобразовании чисел в дату, используется тот факт, что даты в системе хранятся как числа с плавающей точкой, причем целая часть рассматривается как число дней, прошедших с 1/01/1900 (отрицательные значения тоже допустимы), а дробная – как время.

Синтаксис

CDate(value)

Аргументы

value – выражение для преобразования.

Пример

```
Dim res
res = CDate("Окт,12,2000") ' вернет 12/10/2000
res = CDate("12/10")   ' вызовет ошибку
```

Функции CDbl, CSng

Функции преобразовывают выражение в число с плавающей точкой. Функция `CDbl()` преобразовывает аргумент в число двойной точности (`Double`), а функция `CSng()` в число обычной точности (`Single`).

Если аргумент нельзя интерпретировать как число, происходит ошибка выполнения. Для того, чтобы избежать ошибок выполнения, можно воспользоваться функцией `IsNumeric()`, проверяющей, можно ли рассматривать нечто как число.

Синтаксис

```
CSng(value)
CDbl(value)
```

Аргументы

`value` – выражение для преобразования.

Пример

```
Dim res
res = CDbl("122.23") ' вернет 122.23
res = CSng(256)      ' вернет 256.00
```

Функции CInt, CLng

Функции преобразовывают выражение в число с фиксированной точкой. Функция `CLng()` преобразовывает аргумент в длинное целое (`Long`), а функция `CInt()` в короткое целое (`Integer`).

Если число имеет дробную часть, то оно округляется. Имеются функции `Fix()` и `Int()`, которые отбрасывают дробную часть числа при преобразовании.

Если аргумент нельзя интерпретировать как число, происходит ошибка выполнения. Для того, чтобы избежать ошибок выполнения, можно воспользоваться функцией `IsNumeric()`, проверяющей, можно ли рассматривать нечто как число.

Синтаксис

```
CInt(Value)
CLng(Value)
```

Аргументы

`Value` – выражение для преобразования.

Пример

```
Dim res
res = CInt("122.5") ' вернет 122 (до четного!)
res = CLng("121.5") ' вернет 122 (до четного!)
```

Функция VarType

Функция возвращает значение типа переменной `Variant`. Возвращаемое значение представляет собой одну из констант, описывающих тип переменной.

Синтаксис

```
VarType(value)
```

Аргументы

`value` – выражение, тип которого нужно получить.

Пример

```
Dim res
res = VarType(CCur(12)) ' вернет 6 (vbCurrency)
```

Математические функции

В языке *VBScript* имеется полный набор математический функций. Их перечень приведен в следующей таблице.

Функция	Описание
`Abs`	Возвращает абсолютное значение числа.
`Sgn`	Возвращает знак числа.
`Exp, Log`	Возвращают экспоненту или логарифм числа.
`Sin,Cos,Tan,Atn`	Тригонометрические функции.
`Sqr`	Квадратный корень.
`Round`	Округляет число.
`Fix, Int`	Округляет число до целого.
`Rnd`	Возвращает случайное число.
`Round2`	Округляет число традиционным способом.

Функции `Exp`, `Log`, `Sqr` и тригонометрические работают обычным образом и не имеют каких либо особенностей. Функция `Rnd` возвращает случайное число в диапазоне от `0` до `1`. За более подробной информацией об этих функциях обращайтесь к электронной справочной системе. Остальные функции будут рассмотрены более подробно.

Обратите внимание на то, что функция Round2 не является стандартной в языке *VBScript*. Ее можно использовать только в приложениях *Акцент*.

Функция Abs

Функция просто возвращает абсолютное значение (модуль) числа. Если аргумент равен Null, возвращается Null.

Синтаксис

```
Abs(Arg)
```

Аргументы

Arg – любое числовое выражение.

Пример

```
Dim val
val = Abs(22)   ' равно 22
val = Abs(-22) ' равно 22
```

Функция Sgn

Функция возвращает 1, если число положительно, –1, если оно отрицательно и 0, если оно равно нулю. Если аргумент равен Null, возвращается Null.

Синтаксис

```
Sgn(Arg)
```

Аргументы

Arg – любое числовое выражение.

Пример

```
Dim val
val = Sgn(22)   ' равно  1
val = Sgn(-22) ' равно -1
```

Функции Fix и Int

Функции возвращают аргумент, округленный до целого значения. Функции просто отбрасывают дробную часть. Различие в их поведении проявляется при работе с отрицательными числами. Int возвращает первое отрицательное число, меньшее или равное аргументу, а Fix большее или равное аргументу.

Синтаксис

```
Fix(Arg)
Int(Arg)
```

Аргументы

`Arg` – любое числовое выражение.

Пример

```
Dim val
val = Int(-11.6)   ' равно -12
val = Fix(-11.6)   ' равно -11
```

Функция Round

Функция возвращает округленное значение аргумента. Функция работает по «банкирскому» принципу, то есть округляет 0.5 до ближайшего ЧЕТНОГО целого. В программах *Акцент* имеется функция `Round2()`, которая работает традиционным образом.

Если аргумент равен `Null`, функция возвращает `Null`.

Синтаксис

```
Round(Arg [,Sign])
```

Аргументы

`Arg` – любое числовое выражение.

`Sign` – количество знаков после точки. Если не указано, число округляется до целого.

Пример

```
Dim val
val = Round(11.5)   ' равно 12
val = Round(10.5)   ' равно 10
```

Функция Round2

Функция возвращает округленное значение аргумента.

В отличие от стандартной функции `Round`, она работает традиционным способом, то есть округляет 0.5 в большую сторону.

Функция не является стандартной в *VBScript* и доступна только в программах *Акцент*.

Синтаксис

```
Round2(Arg [,Sign])
```

Аргументы

`Arg` – любое числовое выражение.

`Sign` – количество знаков после точки. Если не указано, число округляется до целого.

Пример

```
Dim val
val = Round2(11.5)  ' равно 12
val = Round2(10.5)  ' равно 11
```

Работа с массивами

В языке *VBScript* имеется несколько встроенных функций для работы с массивами. Они приведены в следующей таблице.

Функция	Описание
`Array`	Создает массив из списка аргументов
`LBound, UBound`	Возвращают минимально и максимально возможный индекс массива
`IsArray`	Проверяет, является ли переменная массивом

Функция Array

Функция создает массив из своего списка аргументов. Массив, который вернет эта функция, будет представлять собой переменную типа `Variant`, но не встроенный массив языка *VBScript*. Обратите внимание, что массив, созданный функцией `Array`, может содержать данные любых типов, в том числе и массивов.

Синтаксис

```
Array(Arg_list)
```

Аргументы

`Arg_list` – произвольный список любых значений через запятую.

Пример

```
Dim arr
' массив из 5 элементов
arr = Array(1, 2, 3, 4, 5)
' массив из 3-х элементов,
```

```
' третий сам является массивом
arr = Array(1, 2, Array(3, 4))
```

Функции LBound, UBound

Эта пара функций возвращает максимальное и минимальное значение индекса массива.

Может показаться, что функция `LBound` всегда будет возвращать значение 0, поскольку массивы в *VBScript* индексируются с нуля. Это верно лишь отчасти. Действительно, для массивов, объявленных в программе, нижнее значение границы всегда нулевое. Но, для массивов, которые были получены вызовом методов объектов, нижнее значение границ массива может быть произвольным.

Синтаксис

```
LBound(Arr [,Dim]) ' минимальный индекс
UBound(Arr [,Dim]) ' максимальный индекс
```

Аргументы

`Arr` – имя массива или переменной типа Variant, которая содержит массив.

`Dim` – необязательный аргумент, указывающий на номер измерения массива. Если аргумент не указан, то он считается равным 1.

Пример

```
Dim arr
Dim val
arr = Array(1,2,3) ' массив их 3-х элементов
val = LBound(arr)  ' вернет 0
val = UBound(arr)  ' вернет 2
```

Функция IsArray

Функция просто проверяет, является ли ее аргумент массивом, и возвращает `True`, если это так и `False` в противном случае.

Функция работает как для массивов языка, так и для переменных типа `Variant`, содержащих массив.

Синтаксис

```
IsArray(Array)
```

Аргументы

`Array` – переменная, которую нужно проверить.

Пример

```
Dim arr
' массив из 5 элементов
arr = Array(1, 2, 3, 4, 5)
MsgBox IsArray(arr) ' выведет True
```

Работа с объектами

В языке *VBScript* имеется несколько обобщенных функций для работы с объектными переменными. Эти функции приведены в следующей таблице.

Функция	Описание
`IsObject`	Проверяет, является ли переменная ссылкой на объект
`CreateObject`	Создает внешний ActiveX объект
`GetObject`	Создает внешний ActiveX объект на основе содержимого файла
`LoadPicture`	Загружает картинку из файла и создает стандартный объект `Picture`

Функция IsObject

Функция возвращает `True`, если переменная является ссылкой на какой-либо объект. Если переменная не ссылается никуда (то есть равна `Nothing` то функция все равно вернет `True`.

Синтаксис

```
IsObject(Arg)
```

Аргументы

`Arg`– произвольная переменная.

Функция CreateObject

Функция создает внешний объект и возвращает ссылку на него. Не забывайте, что при работе с объектами использование ключевого слова `Set` обязательно.

Синтаксис

```
Set obj = CreateObject(Class)
```

Аргументы

`Class` –класс объекта. Указывается в виде
`server.object[.version]`.
Например: `"Excel.Worksheet.5"`
`server` – имя приложения, которое создает объект (например `Excel`)
`object` – имя или класс объекта внутри приложения (например `Worksheet`)
`version` – (необязательно) версия приложения

Каждый сервер предоставляет, по крайней мере, один класс объектов. Например, электронная таблица `Excel` предоставляет объект приложения (`Application`), объект рабочего листа (`Worksheet`), объект диаграммы (`Chart`) и т. д.

Примечание. Все внутренние объекты приложений семейства *Акцент* создаются изнутри приложения. Если программа работает внутри приложения *Акцент*, то использование функции `CreateObject` может быть необходимо только для создания внешних объектов.

Пример

```
' Пример создания объекта Excel
Dim sh
' запуск Excel и создание рабочей книги
Set sh = CreateObject("Excel.Workbook")
```

Функция GetObject

Функция создает внешний объект, загружает в него информацию из указанного файла и возвращает ссылку на него. Не забывайте, что при работе с объектами использование ключевого слова `Set` обязательно.

Синтаксис

```
Set obj = GetObject(File [,Class])
```

Аргументы

`File` – полное имя к файлу, содержащему данные.
`Class` – необязательно. Класс объекта. Указывается в виде
`server.object[.version]`.
Например: `"Excel.Worksheet.5"`
`server` – имя приложения, которое создает объект

(например Excel)
object – имя или класс объекта внутри приложения
(например Worksheet)
version – (необязательно) версия приложения.
Если класс объекта не указан, то он определяется по расширению файла.

Примечание. Некоторые приложения позволяют выделить внутри файла отдельные объекты. В этом случае имя объекта указывается после имени файла и отделяется от него восклицательным знаком.

Пример

```
' Пример получения объекта Excel.Worksheet
Dim sh
Set sh = GetObject("c:\MySheet.xls!Sheet1")
```

Функция LoadPicture

Функция загружает картинку из файла и возвращает ссылку на стандартный объект Picture. Ссылкой на объект можно воспользоваться для установки свойства Picture. Можно использовать файлы типов .bmp, .ico, .gif, .jpg и некоторые другие.

Синтаксис

```
LoadPicture(FileName)
```

Аргументы

FileName– полное имя файла с изображением. Обратите внимание, что при поиске файла информация о расположении файлов не принимается в расчет.

Пример

```
' Пример загрузки картинки
Dim pic
Set pic = LoadPicture("c:\accent7.4\logo.bmp")
```

Обработка строк

Язык *VBScript* поддерживает богатый набор встроенных функций, связанных с обработкой текстовых строк.

Перечень функций VBScript, предназначенных для обработки строк, приведен в следующей таблице (в алфавитном порядке).

Функция	Описание
Asc	Возвращает ASCII код первого символа строки
Chr	Возвращает строку из одного символа, ASCII код которого задается аргументом
CStr	Преобразовывает аргумент любого типа в строку
Filter	Выполняет поиск в массиве строк.
Hex,Oct	Преобразовывают аргумент в 16-ричное или 8-ричное представление.
InStr, InStrRev	Ищут в строке образец и возвращает позицию первого или последнего вхождения образца.
Join	Возвращает строку, которая является конкатенацией элементов одномерного массива
LCase,UCase	Преобразовывают строку в нижний или верхний регистр
Left,Right	Возвращают левую или правую часть строки
Len	Возвращает длину строки
LTrim,RTrim, Trim	Удаляют из строки лидирующие пробелы (слева и справа или с обеих сторон)
Mid	Возвращает подстроку из середины строки
Replace	Заменяет одну подстроку на другую
Space, String	Возвращают строку, состоящую из заданного числа пробелов или заданных символов.
Split	Разбивает строку на подстроки и помещает их в массив.
StrComp	Возвращает результат сравнения двух строк
StrReverse	Переставляет символы строки в обратном порядке

В программах *Акцент* предусмотрены дополнительные функции для работы со строками. Их перечень приведен в следующей таблице.

Функция	Описание
CharAt	Возвращает один символ строки по индексу
Spell	Возвращает число прописью
SpellMoney, SpellMoney2	Возвращают сумму прописью. Используют настройки программы или аргументы самой функции.
Token	Извлекает из строки элемент, рассматривая строку как список элементов, разделенных заданным символом.

Далее приведено краткое описание каждой функции с примерами применения.

Функция Asc

Функция возвращает ASCII код первого символа строки. Если аргумент равен `Null` или это пустая строка, происходит ошибка выполнения программы. Обратите внимание, что строчные и прописные функции имеют разные коды. Имеется функция `Chr()`, выполняющая обратное действие.

Синтаксис

```
Asc(String)
```

Аргументы

`String` – строковое выражение. Значимым является только первый символ, код которого возвращается.

Пример

```
Dim code
code = Asc("Строка") ' вернет 209 (код буквы С)
code = Asc("%") ' вернет 37
```

Функция Chr

Функция возвращает строку, состоящую из одного символа, ASCII-код которого задается в качестве аргумента функции. Если аргумент равен `Null`, происходит ошибка выполнения программы. Существует функция `Asc`, выполняющая обратное действие.

Синтаксис

```
Chr(Code)
```

Аргументы

`Code` – ASCII код символа для преобразования.

Пример

```
Dim char
char = Chr(209) ' вернет "С"
char = Chr(37)  ' вернет "%"
char = Chr(10)  ' вернет символ vbLf
```

Функция CStr

Функция преобразовывает аргумент в текстовую строку.

Синтаксис

```
CStr(Arg)
```

Аргументы

`Arg` – выражение любого типа

В зависимости от типа аргумента, функция использует разные алгоритмы преобразования.

Если аргумент типа:	То возвращается:
`Boolean`	`"True"` или `"False"`.
`Date`	Дата в коротком формате с учетом установок языка.
`Null`	Возникает ошибка выполнения программы «Недопустимое использование Null».
`Empty`	Пустая строка ("").
`Error`	Строка `"Error ##"`, где ## - номер ошибки.
`Long`, `Integer`	Строковое изображение числа.
`Single`, `Double`	Строковое изображение числа с учетом системных установок языка для чисел.
`Currency`	Строковое изображение суммы с учетом системных установок языка для денежных значений.

Пример

```
Dim str
str = CStr(22)    ' вернет "22"
str = CStr(True)  ' вернет "True"
```

Примечание. При преобразовании дат, чисел и сумм используются системные установки *Windows*. При преобразовании булевых значений текст будет выводиться на языке локализованной версии системы. Другими словами, в русской версии *Windows* вызов `CStr(True)` вернет значение "Истина".

Функции LTrim, RTrim, Trim

Эта группа функций предназначена для удаления незначащих пробелов, табуляций и символов перевода строки слева, справа или с обеих сторон строки.

Синтаксис

```
LTrim(String) ' удаляет пробелы слева
```

```
RTrim(String) ' удаляет пробелы справа
Trim (String) ' удаляет пробелы слева и справа
```

Аргументы

`String` – строковое выражение.

Пример

```
Dim result
result = LTrim("   Строка") ' вернет "Строка"
result = RTrim("Строка   ") ' вернет "Строка"
result = Trim (" Строка  ") ' вернет "Строка"
```

Функция Filter

Функция выполняет поиск в массиве строк по заданному критерию и возвращает массив найденных элементов. Если совпадений не найдено, функция возвращает пустой массив. Поиск в многомерных массивах не поддерживается.

Синтаксис

```
Filter(Array, Value [,Include][,Compare])
```

Аргументы

`Array` – массив строк, в котором будет выполняться поиск.

`Value` – строковое значение, которое нужно найти.

`Include` – необязательный аргумент. Если он равен `True`, то ищутся строки, содержащие указанное значение `Value`. Если `False` – то ищутся строки, не содержащие указанное значение.

`Compare` – необязательный аргумент, определяющий способ сравнения строк. Подробности в справочной системе.

Пример

```
Dim arr
Dim result

arr = Array("one","two","three")
result = Filter(arr, "o")
' вернет массив строк,
' в которых есть буква "o"("one","two")
```

Функции Hex и Oct

Функция возвращает строку, представляющую собой изображение аргумента в 16-ричном (Hex) или 8-ричном (Oct) представлении. Префиксы &H и &O к строке не добавляются. Если число не целое, то оно предварительно округляется.

Синтаксис

```
Hex(Value)
Oct(Value)
```

Аргументы

Value – числовое выражение для преобразования.

Пример

```
Dim str
str = Hex(209) ' вернет "D1"
str = Oct(209) ' вернет "321"
```

Функции InStr и InStrRev

Функции ищут в строке первое (InStr) или последнее (InStrRev) вхождение подстроки.

Синтаксис

```
InStr([Index,] String, Substr [,Compare])
InStrRev(String, Substr [,Index][,Compare])
```

Аргументы

String – строка для поиска.

Substr – подстрока, которая ищется.

Index – необязательный аргумент, определяющий индекс символа, с которого нужно начинать поиск.

Compare – необязательный аргумент, определяющий способ сравнения строк. Подробности в справочной системе.

Пример

```
Dim ind

ind = InStr("this is a string","s")     ' =4
ind = InStrRev("this is a string","s") ' =11
```

Функция Join

Функция возвращает строку, которая является конкатенацией строк, хранимых в одномерном массиве. Имеется функция Split (стр.69), выполняющая обратное действие.

Синтаксис

```
Join(Array [,Delim])
```

Аргументы

Array – одномерный массив строк.

Delim – необязательный аргумент. Определяет символ, который будет считаться разделителем (будет вставляться между элементами массива). Если не указан, считается равным пробелу.

Пример

```
Dim str
str = Join(Array("one","two","tree"))
' вернет строку "one two three"
str = Join(Array("one","two","tree"),":")
' вернет строку "one:two:three"
```

Функции LCase и UCase

Функции преобразовывают все символы строки в нижний (LCase) или верхний (UCase) регистр. Если строка равна Null, функция возвращает Null.

Синтаксис

```
LCase(String)
UCase(String)
```

Аргументы

String – строка для преобразования.

Пример

```
Dim str
str = LCase("The String") ' вернет "the string"
str = UCase("The String") ' вернет "THE STRING"
```

Функции Left и Right

Функции выделяют из строки подстроку, содержащую заданное количество символов слева (`Left`) или справа (`Right`). Если строка равна `Null`, функция возвращает `Null`.

Синтаксис

```
Left(String, Chars)
Right(String, Chars)
```

Аргументы

`String` – исходная строка.

`Chars` – количество символов, которые нужно извлечь. Если оно равно нулю, возвращается пустая строка, если больше длины строки, то возвращается вся исходная строка.

Пример

```
Dim str
str = Left("The String", 3)  ' вернет "The"
str = Right("The String", 6) ' вернет "String"
```

Функция Len

Функция возвращает длину строки в символах.

Синтаксис

```
Len(String)
```

Аргументы

`String` – строковое выражение.

Пример

```
Dim length
length = Len("The String") ' вернет 10
```

Функция Mid

Функция возвращает подстроку, извлеченную из средней части строки.

Синтаксис

```
Mid(String,Index [,Length])
```

Аргументы

`String` – строковое выражение.

Index – позиция начального символа в строке.

Length – необязательный аргумент. Длина подстроки, которую нужно извлечь. Если не указана или больше длины исходной строки, то возвращается все до конца строки.

Пример

```
Dim result
result = Mid("My string", 4, 3) ' вернет "str"
```

Функция Replace

Функция заменяет все вхождения подстроки в исходной строке на другую подстроку. Строка замены может быть пустой. Тогда функция просто удаляет из исходной строки все вхождения исходной строки.

Синтаксис

```
Replace(Str, What ,Rep [,Index][,Num][,Compare])
```

Аргументы

Str – исходная строка для поиска и замены.

What –строка, которую нужно найти и заменить.

Rep – строка, на которую будет выполняться замена.

Start – индекс в исходной строке, с которого будет начинаться поиск и замена.

Num –(необязательно) максимальное число замен. Если не указано, то заменяются все вхождения.

Compare – необязательный аргумент, определяющий способ сравнения строк. Подробности в справочной системе.

Пример

```
Dim result
result = Replace("a.b.c", ".", "|")
         ' вернет "a|b|c"
```

Функция Space

Функция возвращает строку, состоящую из заданного количества пробелов. Имеется функция String (стр. 70), которая возвращает строку, состоящую из произвольных одинаковых символов.

Синтаксис

```
Space(Count)
```

Аргументы

Count – количество пробелов в результате.

Пример

```
Dim result
result = Space(5) ' вернет строку "     "
```

Функция Split

Функция «расщепляет» строку на части и возвращает одномерный массив подстрок. Имеется функция Join (стр.66), которая выполняет обратное действие, – объединяет массив в одну строку.

Синтаксис

```
Split(String [,Delim] [,Number][,Compare])
```

Аргументы

String – строка для разбиения.

Delim – (необязательно) символ, который будет считаться разделителем. Если аргумент не указан, считается равным пробелу.

Number – (необязательно) максимальное количество подстрок, которые будут извлекаться из строки. Если аргумент не указан, то извлекаются все строки.

Compare – необязательный аргумент, определяющий способ сравнения строк. Подробности в справочной системе.

Пример

```
Dim result
result = Split("a b c")
         ' вернет массив ("a","b","c")
```

Функция StrComp

Функция сравнивает две строки. Функция возвращает **–1**, если первая строка меньше второй, **+1** – если больше и **0** если они равны. Обычно функция используется для сортировки строк. Если один из аргументов равен Null, строка возвращает Null.

Синтаксис

```
StrComp(String1, String2 [,Compare])
```

Аргументы

String1 – первая строка для сравнения.

String2 – вторая строка для сравнения.
Compare – необязательный аргумент, определяющий способ сравнения строк. Подробности в справочной системе.

Пример

```
Dim result
result = StrComp("a","b") ' вернет -1
result = StrComp("b","a") ' вернет +1
```

Функция String

Функция возвращает строку, состоящую из заданного количества одинаковых символов. Имеется функция Space (стр. 68), которая возвращает строку, состоящую из заданного числа пробелов.

Синтаксис

```
String(Count,Code)
```

Аргументы

Count – количество символов в результате.
Code – ASCII-код символа.

Пример

```
Dim result
result = String(5,Asc("a")) ' вернет "aaaaa"
```

Функция StrReverse

Функция «переворачивает» строку (то есть первый символ станет последним, а последний первым). Если аргумент равен Null, возвращается Null.

Синтаксис

```
StrReverse(String)
```

Аргументы

String –строковое выражение.

Пример

```
Dim result
result = StrReverse("abcd") ' вернет "dcba"
```

Функции Акцент для обработки строк

Для более удобной работы со строками в программах семейства *Акцент* предусмотрены некоторые дополнительные функции. Обратите внимание, что этих функций нет в самом языке *VBScript*. Они есть только в программах семейства *Акцент*.

Функция CharAt

Функция возвращает строку, состоящую из одного символа, выбранного из исходной строки по заданному индексу. Функция аналогична `Mid` с длиной строки равной 1. В основном используется для совместимости с предыдущими версиями *Акцент*.

Синтаксис

```
CharAt(String,Index)
```

Аргументы

`String` – исходная строка.
`Index` – номер выбираемого символа (начиная с 1).

Пример

```
Dim res
res = CharAt("abcd",2) ' вернет "b"
res = Mid("abcd",2,1)  ' вернет "b"
```

Функция Spell

Функция возвращает целое число прописью. Для работы с суммами используются функции `SpellMoney` и `SpellMoney2`.

Синтаксис

```
Spell(Number [,Lang][,Woman])
```

Аргументы

`Number` – числовое выражение. Если дробная часть не равна нулю, число округляется.
`Lang` – (необязательно) Язык, на котором выводится текст. Если аргумент равен нулю или не указан – русский, если равен **1** – украинский.
`Woman` – (необязательно) Род для числа. `False` (по умолчанию) – мужской, `True` – женский.

Пример

```
Dim str
str = Spell(55) ' вернет "пятьдесят пять"
str = Spell(1, ,False) ' вернет "один"
str = Spell(1, ,True)  ' вернет "одна"
```

Функции SpellMoney, SpellMoney2

Функция возвращает сумму прописью. Функция `SpellMoney` берет наименование денежных единиц из свойств валюты, указанных в базе данных. Для функции `SpellMoney2` наименование денежных единиц задается в аргументах функции.

Синтаксис

```
SpellMoney(Sum [,CurID][,Lang][,Mode])
SpellMoney2(Sum [,Descr][,Lang][,Mode])
```

Аргументы

`Sum` – выражение типа `Currency`. Если дробная часть имеет более 2-х знаков, число округляется.

`CurID` – (необязательно) Идентификатор (`ID`) валюты. Если аргумент не указан, то берется базовая валюта (ее `ID` равен **1**).

`Descr` – (необязательно) Описание наименования денежных единиц. Подробности – в справочной системе.

`Lang` – (необязательно) Язык, на котором выводится текст. Если аргумент равен нулю или не указан – русский, если равен единице – украинский.

`Mode` – (необязательно) Режим вывода копеек. Если аргумент не указан или равен нулю, копейки выводятся цифрами, если равен единице – словами.

Пример

```
Dim str
str = SpellMoney(5.00)
     ' вернет " пять грн. 00 коп"

str = _
SpellMoney2(5.25,"1|руб|руб|руб|0|коп|коп|коп")
     ' вернет " пять руб 25 коп"
```

Функция Token

Функция рассматривает строку, как список элементов, разделенных заданным символом. Функция возвращает элемент по его номеру. Если номер больше количества элементов, просто возвращается пустая строка.

Синтаксис

```
Token(String,Number [,Delim])
```

Аргументы

`String` – исходная строка.

`Number` – номер выбираемой подстроки (начиная с 1).

`Delim` – (необязательно) Разделитель. Если не указан, то в качестве разделителя используется запятая.

Пример

```
Dim res
res = Token("a,b,c",2)      ' вернет "b"
res = Token("a b c",2," ") ' вернет "b"
```

Работа с датами

Даты и время являются важными элементами языка. Для переменных типа `Date` определены операции присваивания (=) и сравнения (`>, <, =, >=, <=`).

Обратите внимание, что с датами и временем нельзя выполнять арифметические операции. Нельзя, например, сложить две даты или вычесть одну из другой. Для выполнения таких действий служат специальные встроенные функции, например, `DateAdd` для добавления к дате или вычитания из даты заданного интервала времени или `DateDiff`, для вычисления количества интервалов между двумя датами.

Примечание. Даты в языке *VBScript* хранятся в виде чисел с плавающей точкой, в которых целая часть представляет собой количество дней, прошедших с 1.01.0100, а время – число интервалов, начиная с 00:00 часов. Интервал подобран так, что 23:59:59.9999 будет равно 0.999999(9). Таким образом, сравнение дат превращается в достаточно тривиальную операцию.

Встроенные функции

Для операций с датами и временем в языке *VBScript* имеется большой набор встроенных функций. Перечень таких функций (в алфавитном порядке) приведен в следующей таблице.

Функция	Описание
`CDate`	Преобразовывает аргумент в дату. Функция описана в разделе, посвященном преобразованиям типов (стр. 51).
`Date,Time,Now`	Возвращают текущую дату и время.
`DateAdd`	Прибавляет к дате интервал.
`DateDiff`	Возвращает количество интервалов между двумя датами.
`DatePart`	Возвращает заданный элемент даты.
`DateSerial`	Создает переменную типа `Date` по году, месяцу и дню.
`DateValue`	Преобразовывает строку в дату.
`Day, Month, Year`	Возвращают день месяц и год для даты.
`FormatDateTime`	Форматирует дату и/или время.
`Hour, Minute, Second`	Возвращают часы, минуты и секунды времени.
`IsDate`	Проверяет, можно ли преобразовать аргумент в дату.
`MonthName`	Возвращает наименование месяца.
`Timer`	Возвращает количество секунд, прошедших с 00:00 часов.
`TimeSerial`	Создает переменную типа `Date` и устанавливает в ней время (час:мин:сек).
`TimeValue`	Преобразовывает строку во время.
`Weekday`	Возвращает номер дня недели.
`WeekdayName`	Возвращает наименование дня недели.

В программах *Акцент* предусмотрены дополнительные функции, используемые для работы с датами. Их перечень приведен в таблице ниже (в алфавитном порядке).

Функция	Описание

`FirstDayInMonth`
`LastDayInMonth`
`FirstDayInQuarter`
`LastDayInQuarter`
	Возвращают дату первого или последнего дня месяца или квартала.

`FormatDate2`	Форматирует дату и/или время
`TrimTime`	Возвращает дату с временем, равным 00:00:00.

Функции Date, Time, Now

Эта группа функций предназначена для доступа к текущей системной дате и/или времени. Возвращаемое значение имеет тип `Date`.

Синтаксис

```
Date() ' возвращает дату
Time() ' возвращает время
Now()  ' возвращает дату и время
```

Пример

```
если сегодня 26/03/2000 12:45:33, то
Date() вернет 26/03/2000
Time() вернет 12:45:33
Now()  вернет 26/03/2000 12:45:33
```

Функция DateAdd

Функция возвращает дату, к которой добавлен заданный интервал. Интервал может быть как положительным (тогда результат позже), так и отрицательным (тогда результат раньше).

Синтаксис

```
DateAdd(What, Number, Date)
```

Аргументы

`What` – текстовая строка, задающая вид интервала, который будет добавлен к дате. Возможные значения приведены ниже.

`Number` – число интервалов типа `What`, которые нужно добавить. Отрицательное значение означает, что число интервалов нужно вычесть, а не добавить.

`Date` – дата, к которой добавляются интервалы.

Возможные значения интервалов:

Строка	Интервал
"yyyy"	год
"q"	квартал
"m"	месяц
"d"	день
"ww"	неделя
"h"	час
"n"	минута
"s"	секунда

Пример

```
Dim result
result = DateAdd("m",1,#1999/3/26#)
         ' вернет 26/04/1999

result = DateAdd("yyyy",1,#1999/3/26#)
' вернет 26/03/2000
```

Функция DateDiff

Функция возвращает количество интервалов между двумя датами. Если `Date1` хронологически позже `Date2`, то функция возвращает отрицательное значение.

Синтаксис

```
DateDiff(What, Date1, Date2 [,Week][,YearDay])
```

Аргументы

`What` – текстовая строка, задающая вид интервала, который будет добавлен к дате. Возможные значения приведены ниже.

`Date1` – первая дата.

`Date2` – вторая дата.

`Week, YearDay` – необязательные аргументы, задающие день, с которого начинается неделя, а также, какую неделю считать первой в году. Обычно задаются специальными константами с префиксом `vb`. Подробности в электронной справочной системе.

Возможные значения интервалов:

Строка	Интервал
"yyyy"	год
"q"	квартал
"m"	месяц
"d"	день
"y"	день года
"w"	день недели
"ww"	неделя
"h"	час
"n"	минута
"s"	секунда

Пример

```
Dim result
result = DateDiff("d",#1999/3/26#,#2000/3/26#)
' вернет 366 (дней) (2000 год – високосный)

result = DateDiff("ww",#1999/3/26#,#2000/3/26#)
' вернет 52 (недель)
```

Функция DatePart

Функция возвращает заданный элемент даты.

Синтаксис

```
DatePart(What, Date [,Week][,YearDay])
```

Аргументы

`What` – текстовая строка, задающая вид интервала, который будет добавлен к дате. Возможные значения такие же, как и в функции `DateDiff` (стр. 76).

`Date` – дата, элемент которой нужно получить.

`Week, YearDay` – необязательные аргументы, задающие день, с которого начинается неделя, а также, какую неделю считать первой в году. Подробности в электронной справочной системе.

Пример

```
Dim result
result = DatePart("d",#1999/3/26#)  ' вернет 26
result = DatePart("ww",#1999/3/26#) ' вернет 13
```

Функция DateSerial

Функция возвращает дату, построенную из заданного дня, месяца и года. Можно задавать относительные даты. Например, вызов функции в виде `DateSerial(1999,13,1)` вернет `01/01/2000`. Для того чтобы получить полную дату, включая время, просто прибавьте к дате время, полученное функцией `TimeSerial`.

Синтаксис

```
DateSerial(Year, Month, Day)
```

Аргументы

`Year` – год от 0 до 9999.

`Month` – месяц (не обязательно от 1 до 12).

`Day` – день (не обязательно от 1 до 31).

Пример

```
Dim result
result = DateSerial(1999,3,26)
' равно 26/03/1999
```

Примечание. Если год указан двумя цифрами, то все значения от 30 до 99 интерпретируются как даты в XX веке (1930-1999 гг.), а значения от 0 до 29 – как даты в XXI веке (2000-2029 гг.)

Функция DateValue

Функция пытается преобразовать строку в дату, используя системные установки *Windows*. Можно задавать даты в коротком или длинном формате. Если год пропущен, берется текущий год. Если указано время, то оно разбирается, но не используется.

Синтаксис

```
DateValue(String)
```

Аргументы

`String` – текстовое представление даты, возможно с временем.

Пример

```
Dim result
result = DateValue("26 Март 1999")
' равно 26/03/1999
```

Функции Day, Month, Year

Эта группа функций предназначена для «разбиения» даты на составные части – день, месяц и год.

Функция `DateSerial` выполняет обратную операцию: «сборку» даты из значений дня, месяца и года.

Синтаксис

```
Day  (Date) ' возвращает день
Month(Date) ' возвращает месяц
Year (Date) ' возвращает год
```

Аргументы

`Date` – выражение типа `Date`. Если аргумент нельзя преобразовать в дату, происходит ошибка выполнения программы. Чтобы избежать ошибок, можно использовать функцию `IsDate()`.

Пример

```
Dim result
Dim value
value = #1999/03/26#
result =   Day(value) ' вернет 26
result = Month(value) ' вернет 03
result =  Year(value) ' вернет 1999
```

Функция FormatDateTime

Функция возвращает текстовое представление даты и/или времени с использованием текущих системных установок *Windows*.

В программы *Акцент* встроена функция `FormatDate2` (стр. 81), которая форматирует дату независимо от системных установок.

Синтаксис

`FormatDateTime``(Date [,Format])`

Аргументы

`Date` – дата для форматирования.

`Format` – вид формата. Задается следующими стандартными константами:

Константа	Значение	Описание
`vbGeneralDate`	0	Дата и/или время. Если это только дата, то она выводится в коротком формате. Если это только время, то выводится время в длинном формате. Если это и то и другое – выводится все.
`vbLongDate`	1	Дата в длинном формате
`vbShortDate`	2	Дата в коротком формате
`vbLongTime`	3	Время в длинном формате
`vbShortTime`	4	Время в коротком формате

Пример

```
Dim str
str = FormatDateTime(#1999/03/26#, vbLongDate)
' в русской Windows вернет "26 марта 1999г."
```

Функции Hour, Minute, Second

Эта группа функций предназначена для «разбиения» времени на составные части – часы, минуты и секунды. Функция `TimeSerial` выполняет обратную операцию: «сборку» времени из значений часов минут и секунд.

Синтаксис

```
Hour  (Time) ' возвращает часы
Minute(Time) ' возвращает минуты
Second(Time) ' возвращает секунды
```

Аргументы

`Time` – выражение типа `Date`.

Пример

```
Dim result
Dim value
value = #1999/03/26 11:33:59#
result =   Hour(value) ' вернет 11
result = Minute(value) ' вернет 33
result = Second(value) ' вернет 59
```

Функция IsDate

Функция проверяет, можно ли безопасно (без ошибок) преобразовать аргумент в дату и/или время. Применение функции позволяет избежать ошибок на этапе выполнения программы.

Синтаксис

```
IsDate(Arg)
```

Аргументы

`Arg` – произвольное выражение.

Пример

```
' Ввод даты пользователем
Dim dt
Do
```

```
    dt = InputBox("Введите дату:")
    If IsDate(dt) Then
       Exit Do
    Else
       MsgBox "Вы ошиблись! Повторите ввод"
    End If
Loop
```

Функция TimeSerial

Функция возвращает время, построенное из заданных часов, минут и секунд. Можно задавать относительные времена. Функция автоматически пересчитывает значения в часы, минуты и секунды. Например, вызов функции в виде `TimeSerial(12,61,00)` вернет `13:01:00`.

Для того чтобы получить полную дату, просто прибавьте к времени дату, полученную функцией `DateSerial`.

Синтаксис

```
TimeSerial(Hour, Minute, Second)
```

Аргументы

`Hour` – часы (не обязательно от 0 до 23).
`Minute` – минуты (не обязательно от 0 до 59).
`Second` – секунды (не обязательно от 0 до 59).

Пример

```
Dim result
result = TimeSerial(11,55,59) ' равно 11:55:59
result = TimeSerial(11,-55,59)' равно 10:06:39
```

Функции Акцент для работы с датами

Для более удобной работы с датами в программах семейства *Акцент* предусмотрены некоторые дополнительные функции.

Обратите внимание, что этих функций нет в самом языке *VBScript*. Они есть только в программах *Акцент*.

Функция FormatDate2

Функция возвращает текстовое представление даты и/или времени в заданном формате.

Синтаксис

FormatDate2(Date, Format)

Аргументы

Date – дата для форматирования.

Format –формат. Задается текстовой строкой. Описание формата приведено ниже.

Строка формата

Строка формата это текстовая строка, содержащая специальные символы, которые заменяются нужными значениями. В начале строки формата может находиться модификатор языка из двух букв (в круглых скобках).

Модификаторы языка:

Текст	Описание
(ru)	русский (по умолчанию)
(ua)	украинский
(en)	английский

Элементы формата:

Элемент	Описание
dd	день месяца (всегда 2 цифры)
mm	месяц года (всегда 2 цифры)
mmm	сокращенное наименование месяца (3 буквы)
mmmm	полное наименование месяца
yy	год без столетия (2 цифры)
yyyy	год со столетием (4 цифры)
hh	часы (всегда 2 цифры)
nn	минуты (всегда 2 цифры)
ss	секунды (всегда 2 цифры)

Пример

```
Dim s
s = FormatDate2(#1999/03/26#, "(ua)dd mmmm yy")
' вернет "26 березня 99"
```

Функции интерфейса пользователя

Функции интерфейса пользователя предназначены для вывода информационных сообщений и для запроса от пользователя некой информации. Перечень этих функций приведен в таблице.

Функция	Описание
`MsgBox`	Выводит окно с информационным сообщением.
`InputBox`	Выводит окно с вопросом, которое позволяет ввести текстовую строку
`Ask`	Выводит запрос с кнопками **[Да]** и **[Нет]**.

Обратите внимание, что функции `Ask` нет в самом языке *VBScript*. Она есть только в программах *Акцент*.

Функция MsgBox

Выводит на экран окно с сообщением и несколькими кнопками выбора. Функция ожидает, пока пользователь нажмет кнопку и возвращает выбор, сделанный пользователем.

Синтаксис

MsgBox(Msg [,Flags][,Caption][,File,Context])

Аргументы

`Msg` – Строка сообщения. Для разбиения сообщения на строки можно использовать символ `vbLf`. Длина строки не должна превышать 1024 символов.

`Flags` – набор флажков, определяющих внешний вид и поведение окна сообщения. Флажки описаны ниже.

`Caption` – заголовок окна сообщения. Если не указан, заголовок будет равен **«VBScript».**

`File, Context` – имя файла справки и номер раздела, который будет выводиться при нажатии клавиши `F1` в окне сообщения.

Элемент `Flags` указывает:

- какие кнопки должны отображаться в окне сообщения,
- какой значок следует отобразить,
- какая кнопка будет активной по умолчанию. Эта кнопка срабатывает, если пользователь нажмет клавишу Enter.

Значение флажков представляет собой сумму нужных значений (максимум по одному из каждой группы).

Возможные значения приведены ниже в следующих таблицах.

Константы для кнопок

Константа	Значение	Описание
vbOKOnly	0	Только кнопка **[OK]** (по умолчанию).
vbOKCancel	1	Кнопки **[OK]** и **[Отмена]**
vbAbortRetryIgnore	2	Кнопки **[Прервать]**, **[Повтор]** и **[Пропустить]**
VbYesNoCancel	3	Кнопки **[Да]**, **[Нет]**, **[Отмена]**
VbYesNo	4	Кнопки **[Да]** и **[Нет]**.
VbRetryCancel	5	Кнопки **[Повтор]**, **[Отмена]**

Константы для значков

Константа	Значение	Описание
VbCritical	16	Значок **«Стоп»**
VbQuestion	32	Значок **«Вопрос»**
VbExclamation	48	Значок **«Внимание»**
VbInformation	64	Значок **«Информация»**

Константы для кнопок «по умолчанию»

Константа	Значение	Описание
vbDefaultButton1	0	Первая (по умолчанию).
vbDefaultButton2	256	Вторая
vbDefaultButton3	512	Третья
vbDefaultButton4	768	Четвертая

Примечание. Если пользователь нажимает клавишу Enter, то срабатывает кнопка по умолчанию (указанная соответствующей константой vbDefaultButtonX). Если пользователь нажимает клавишу Esc, или закрывает окно при помощи кнопки **«закрыть окно»**, то срабатывает кнопка [Отмена].

После завершения работы с окном сообщения функция MsgBox возвращает одно из следующих значений.

Константа	Значение	Описание
VbOk	1	Нажата кнопка **[OK]**.
vbCancel	2	Нажата кнопка **[Отмена]**.
vbAbort	3	Нажата кнопка **[Прервать]**.
vbRetry	4	Нажата кнопка **[Повтор]**.
vbIgnore	5	Нажата кнопка **[Пропустить]**.
vbYes	6	Нажата кнопка **[Да]**.
vbNo	7	Нажата кнопка **[Нет]**.

Функция InputBox

Выводит на экран диалоговое окно, позволяющее ввести текстовую строку. Ожидает завершения ввода и возвращает введенную строку.

Синтаксис

InputBox(Ask[,Caption][,Def],[X,Y][,File,Cntxt])

Аргументы

Ask – Строка вопроса.
Caption – Заголовок окна сообщения.
Def – значение ответа «по умолчанию».
X, Y – координаты верхнего левого угла окна запроса.
File, Context – имя файла справки и номер раздела, который будет выводиться при нажатии клавиши F1 в окне сообщения.

Если указаны файл и контекст, то в окне появится кнопка **[Справка]**.

Если пользователь нажал кнопку **[OK]** или клавишу **Enter**, функция вернет введенное значение. Если пользователь нажал кнопку **[Отмена]** или клавишу Esc, функция вернет пустую строку.

Функция Ask

Функция аналогична MsgBox с кнопками **[Да]** и **[Нет]**. Если нажата кнопка **[Да]**, функция возвращает True, в противном случае – False.

Синтаксис

Ask(Prompt)

Аргументы

Prompt – Строка запроса.

Ссылки на функции

В языке *VBScript* имеется возможность использовать подпрограммы (`Sub`) или функции (`Function`) как объекты. Можно создавать массивы функций, передавать их как параметры и т.д.

Ссылка на функцию в *VBScript* является аналогом «указателя на функцию» в классических языках программирования, таких как C/C++ или аналогом функции «обратного вызова» (callback) в *Windows API*.

Получить ссылку на функцию можно с помощью встроенной функции `GetRef`

Функция	Описание
`GetRef`	Возвращает ссылку на функцию или подпрограмму

Функция GetRef

Возвращает объект, представляющий собой ссылку на функцию. Ссылки на функции обычно используются в универсальных алгоритмах типа сортировки, поиска и т.д.

Синтаксис

```
GetRef(Name)
```

Аргументы

`Name` – Имя функции.

Пример

```
Dim f
Set f = GetRef("Compare")
```

Обратите внимание, что ссылка на функцию фактически представляет собой объект, поэтому применение ключевого слова `Set` обязательно. Еще одной часто распространенной ошибкой является пропуск кавычек в аргументе функции. Аргументом является именно текстовая строка, а не имя функции.

Прочие функции Акцент

В этом описаны полезные дополнительные функции, добавленные в программы *Акцент*. Перечень этих функций приведен в таблице.

Функция	Описание
IIF	Условная функция
CheckNull	Проверка на Null.
DoEvent	Выполняет обработку событий.
App	Возвращает объект приложения (Application)
Meter,WaitCursor	Возвращает объекты WaitCursor и Meter.
QuickSort	Универсальная функция быстрой сортировки.
Workarea	Возвращает объект текущей открытой рабочей области
IsFileExists	Проверяет, существует ли заданный файл

Обратите внимание, что этих функций нет в самом языке *VBScript*. Они имеются только в программах *Акцент*.

Функция IIF

Функция возвращает значение одного из выражений в зависимости от условия. Следует обратить внимание на одну особенность применения этой функции. В отличие от инструкции IF, при выполнении этой функции вычисляются оба выражения. Другими словами, даже если условие Condition не выполняется, выражение FalsePart тоже будет вычислено. Это может привести к нежелательным побочным эффектам, особенно если в выражениях используются вызовы функций. Подобное поведение связано с особенностями интерпретации выражений в языке VBScript.

Синтаксис

```
IIF(Condition,TruePart,FalsePart)
```

Аргументы

Condition – Условие.
TruePart – Значение функции, если условие равно True.
FalsePart – Значение функции, если условие равно False.

Пример

```
Dim s
s = IIF(sum>0, "положительное", "отрицательное")
```

Функция CheckNull

Функция проверяет значение на Null. Если аргумент не равен Null, возвращается проверяемое значение, если равен Null – замещаемое значение. Функция похожа на IsNull языка *T-SQL*.

Синтаксис

```
CheckNull(Check,Replace)
```

Аргументы

Check – проверяемое значение.
Replace – замещаемое значение.

Пример

```
' Замена значений Null на пустую строку
Dim val
val = CheckNull(Field, "")
```

Функция DoEvent

Функция выполняет обработку событий, которые накопились в очереди. Она предназначена для того, чтобы разрешить перерисовку экрана и реакцию на нажатие клавиш мыши в процессе какой-либо длительной обработки.

Синтаксис

```
DoEvent()
```

Обратите внимание. Фактически, эта функция разрешает параллельную обработку кода. Это означает, что, например, форму можно будет закрыть в процессе работы. Для того чтобы избежать неожиданностей, запрещайте закрытие формы или диалога на время выполнения длительных действий с использованием DoEvent.

Функция App

Функция возвращает объект Application текущего приложения. Объект существует всегда и функция всегда завершается успешно. Обратите внимание, что у большинства объектов имеется свойства Application, которое также возвращает объект приложения.

Синтаксис

```
App()
```

Пример

```
Dim app
Set app = App()
```

Функции Meter и WaitCursor

Функции создают и возвращают соответствующий одноименный объект.

Объект `Meter` представляет собой индикатор прогресса, а объект `WaitCursor` устанавливает курсор мыши в виде песочных часов.

Синтаксис

```
Meter()
WaitCursor()
```

Функция QuickSort

Функция представляет собой универсальный алгоритм сортировки. Для того, чтобы отсортировать что-либо, это что-то должно быть похоже на массив, то есть иметь прямую адресацию посредством индексов. Например, можно отсортировать массив или коллекцию.

Сама функция «не знает», что она сортирует. Для связи с тем элементом, который сортируется служат две ссылки на функции – функция сравнения `CompareFunc` и функция перестановки элементов `SwapFunc`.

Функция сравнения должна быть функцией (`Function`) и иметь три аргумента. Имя функции не существенно. Например:

```
Function CompareItems(N1, N2, Param)
  ...
End Function
```

Аргументы `N1` и `N2` задают индексы сравниваемых элементов (начиная с 0), а аргумент `Param` просто передается «насквозь» из функции `QuickSort`. Функция должна вернуть `True`, если первый элемент меньше второго.

Функция перестановки должна быть подпрограммой (`Sub`) и также иметь три аргумента:

```
Sub SwapItems(N1, N2, Param)
  ...
End Sub
```

Аргументы N1 и N2 задают индексы переставляемых элементов (начиная с 0), а аргумент Param просто передается «насквозь» из функции QuickSort. Функция должна переставить местами элементы N1 и N2.

Синтаксис

```
QuickSort(CompareFunc, SwapFunc, Count, Param)
```

Аргументы

CompareFunct – ссылка на функцию сравнения.
SwapFunc – ссылка на функцию.
Count – количество сортируемых элементов.
Param – произвольный параметр, который передается в функцию сравнения.

Пример (сортировка массива)

```
Option Explicit
Dim a, s1, s2
' исходный массив
a = Array(7, 6, 2, 9, 21, 22, 22, 33, 1, -11)
' подпрограмма перестановки элементов
Sub Swap(p1, p2, x)
  Dim t
  t = a(p1)
  a(p1) = a(p2)
  a(p2) = t
End Sub
' функция сравнения элементов
Function LessThan(p1, p2, x)
  LessThan = a(p1) < a(p2)
End Function
' Сохраним исходный массив
s1 = Join(a, ";")
' Сортировка. Кол-во элементов = Ubound + 1
QuickSort GetRef("Less"), GetRef("Swap"), _
          Ubound(a) + 1, 0
' Результат сортировки
s2 = Join(a, ";")

MsgBox "Перед сортировкой: " & s1 & vbLf & _
       "После сортировки: " & s2
```

Функция Workarea

Функция возвращает объект `Workarea` текущей открытой рабочей области. Рабочая область представляет собой текущую открытую базу данных. Если не открыта никакая база данных, функция возвращает `Nothing`.

Синтаксис

```
Workarea()
```

Пример

```
Dim wa
Set wa = Workarea()
```

Функция IsFileExists

Функция проверяет наличие указанного файла. Если файл существует, функция возвращает `True`. Если полный путь к файлу не указан, то используется информация о расположении из файла `.ARD`.

Синтаксис

```
IsFileExists(FileName)
```

Аргументы

`FileName` – Строка, задающая имя файла.

Глава V.

Объекты Акцент

В этой главе подробно описана объектная модель программы *Акцент 7.4*, перечислены все объекты, их методы и свойства, описано их назначение, а также приведены примеры их использования.

Объектная модель по праву считается основой любого программируемого приложения. Программа *Акцент* не является исключением. Все объекты системы (а их более 100) сгруппированы в несколько *библиотек*.

- `Impact Accent 7.4` – основные объекты приложения.
- `Impact Common 7.4` – различные вспомогательные объекты.
- `Impact Reports 7.4` – объекты отчетов.
- `Impact Form 7.4` – работа с формами.
- `Impact Dialog 7.4` – работа с диалогами.
- `Impact Sheet 7.4` – работа с электронными таблицами.

Первые три библиотеки будут рассмотрены в этой главе. Работе с формами, диалогами и электронными таблицами посвящены отдельные главы.

Основные объекты

Основные объекты системы предназначены для работы с бухгалтерскими сущностями, такими как счет, корреспондент, валюта и т.д. Основные объекты находятся в библиотеке `Impact Accent 7.4`. Далее будут рассмотрены объекты, которые находятся в этой библиотеке.

Объект Application

Объект `Application` (приложение) представляет собой текущее приложение. В отличие от рабочей области, которая связана с открытой базой данных, объект `Application` не зависит от базы данных. Он существует

даже тогда, когда никакая база данных не открыта. Это обуславливает назначение свойств и методов этого объекта. К объекту `Application` относится все, что не требует доступа к базе данных.

Многие объекты имеют свойство только для чтения Application, которое возвращает ссылку на текущее приложение. Кроме того, имеется глобальная функция `App()`, которая делает тоже самое.

Примечание: Обратите внимание, что любой код программы на *VBScript* внутри приложения выполняется в контексте текущей базы данных. Может показаться, что объект `Workarea` существует всегда. Однако никто не мешает программно закрыть базу данных. Система останется под управлением кода *VBScript*, а открытой базы данных не будет.

Основные свойства объекта `Application` следующие.

- **`Version`** – версия приложения в виде текстовой строки. Обратите внимание, что объект `Workarea` тоже имеет свойство `Verison`. Однако оно возвращает версию текущей открытой базы данных.
- **`Build`** – целое число, определяющее номер редакции (*build*) приложения. Полезно в том случае, если какие-то программы используют новые возможности, которых не было в предыдущих редакциях.
- **`AppType`** – тип приложения в виде строки текста. Возможные значения `"DAO"` для настольных версий и `"SQL"` для версий клиент-сервер.
- **`Visible`** – установка этого свойства в `False` приводит к тому, что окно приложения становится невидимым. Удобно использовать, если приложение запущено под управлением `OLE-Automation`.
- **`Property`** – возвращает или устанавливает значение произвольного текстового свойства. Свойство идентифицируется по имени. Обратите внимание, что значения этих свойств не сохраняются между сеансами работы с программой. Основное назначение свойства **Property** – передача информации между различными контекстами, например, при работе с Web-интерфейсом программы.

Объект `Application` имеет несколько методов, позволяющих открывать формы, диалоги, таблицы, а также выполнять модули. Для этого служат следующие методы.

- **CreateForm** – открывает заданную форму и, возможно, загружает в нее операцию.
- **CreateDialog** – загружает из файла описание диалога. Обратите внимание, что для вывода диалога на экран используется метод `DoModal` самого диалога.
- **CreateBook** – открывает файл электронной таблицы или создает новую электронную таблицу.
- **RunModule** – загружает и выполняет модуль *VBScript*.
- **ShowDHTMLDialog** – показывает на экране диалог *Dynamic HTML*. Более подробная информация о работе с *HTML* в среде Акцент 7.4 приведена в главе **XII** (страница 266)

Более подробно работа с этими методами описана в электронной справочной системе.

Многоуровневые коллекции

Все основные сущности программы, такие как папки, счета, корреспонденты, объекты учета и другие, организованы в программе в виде многоуровневых (иерархических) коллекций. Такие коллекции часто называются *деревьями*.

Многоуровневость коллекции означает тот факт, что каждый ее элемент содержит такую же коллекцию дочерних элементов. В свою очередь, элемент дочерней коллекции имеет свою дочернюю коллекцию и т.д. Такие структуры данных в программировании называются *рекурсивными*. Обрабатывать такие структуры наиболее удобно рекурсивными функциями (это такие функции, которые прямо или косвенно вызывают сами себя).

Все объекты, организованные в виде деревьев имеют несколько общих свойств, позволяющих их эффективно обрабатывать. Свойства, связанные с многоуровневыми коллекциями таковы:

- **Children** – коллекция вложенных объектов. Коллекция содержит объекты такого же типа, как и родительский объект.
- **HasChildren** - булево значение, если оно равно `True`, то у объекта имеются вложенные объекты. Это свойство имеет особенность. Оно возвращает правильное значение до того, как дочерняя коллекция будет загружена из базы данных. Использование этого свойства позволяет ускорить обработку коллекций.

- **Nested** – коллекция вложенных объектов всех уровней. В отличие от свойства `Children`, возвращает все вложенные объекты до последнего уровня.

Так как все элементы, входящие в многоуровневые коллекции отображаются в деревьях пользовательского интерфейса, они имеют некоторый набор общих свойств.

- **Name** – наименование объекта
- **Tag** – произвольный признак объекта
- **Memo** – примечание к объекту
- **ID** – свойство только для чтения, которое возвращает идентификатор объекта в базе данных.
- **Type** – тип объекта. Диапазон возможных значений зависит от типа объекта. Более подробные сведения приведены в электронной документации.
- **Image** – номер картинки в системном списке, который будет использоваться для отображения элемента соответствующего типа.
- **DrawText** – текстовая строка, которая используется для отображения элемента в дереве. Формат строки зависит от настроек приложения (команда **Сервис | Настройка | Вид**)
- **Params** – свойство возвращает коллекцию параметров. Все объекты многоуровневых коллекций могут содержать параметры.

Пример работы с коллекцией

Предположим, что нужно показать многоуровневую коллекцию корреспондентов в дереве диалога. Для этого воспользуемся элементом управления `DlgTreeView`.

Если попытаться сразу заполнить дерево всеми элементами данных, то становится очевидным, что так делать нельзя. Процесс занимает слишком много времени. Воспользуемся тем, что свойство `HasChildren` правильно показывает наличие вложенных элементов до того, как будет получена коллекция дочерних элементов с помощью свойства `Children`. Фактически, получение значения свойства `HasChildren` не требует дополнительного обращения к базе данных.

Процесс заполнения дерева будет выглядеть следующим образом. Заполняем один уровень дерева (корневой). Для всех элементов, имеющих вложенные (HasChildren=True) добавляем для этого элемента пустой дочерний элемент. Он нужен только для того, чтобы элемент дерева можно было раскрыть.

Далее напишем обработчик события OnExpand, которое возникает при попытке раскрыть узел дерева. В этом обработчике удаляем пустой элемент, а затем снова заполняем этот уровень дерева.

Для того чтобы при сворачивании узла вернуть все в исходное состояние, нужен обработчик события OnCollapse, которое возникает при сворачивании узла дерева. В этом обработчике удаляем все вложенные элементы и снова добавляем пустой элемент для того, чтобы узел можно было снова раскрыть.

Обратите внимание на то, что нам понадобится идентификатор элемента, хранящегося в дереве. Для этого воспользуемся свойством NodeParam, которое позволяет связать с каждым узлом дерева целое число.

Пример программы

```
' ------------------------------------------
' Константы для вставки узлов в дерево
' ------------------------------------------
Const TREE_FIRST  = 1 ' В начало
Const TREE_LAST   = 2 ' В конец
' ------------------------------------------
' Константы для номеров изображений
' ------------------------------------------
Const ROOT_IMAGE  = 3 ' корень дерева
'------------------------------------------
' Эта программа загружает для элемента с заданным ID
' дочерние элементы и вставляет их в узел Node.
'------------------------------------------
Sub FillTree(ID, Node)
  ' Локальные переменные
  Dim Agent    ' Текущий корреспондент
  Dim NewNode  ' Вновь вставленный узел
  Dim i        ' Счетчик
  Dim At       ' Режим вставки в дерево

  ' Получить ссылку на коллекцию корреспондентов
  If ID = 0 Then
```

```
    ' Если ID = 0, то берем коллекцию верхнего уровня
    Set Agent = Workarea.Agents
  Else
    ' Иначе, коллекцию для конкретного корреспондента
    Set Agent = Workarea.Agent(ID).Children
  End If
  ' Для всех элементов коллекции
  For i=1 To Agents.Count
    ' По умолчанию вставляем в конец списка
    At = TREE_LAST
    With Agent(i)
      ' Вставляем узел в дерево
      NewNode = Tree.InsertNode(.Name,.Image,Node,At)
      Tree.NodeParam(NewNode) = .ID
      ' Если есть дочерние, добавим фиктивный узел
      If .HasChildren Then
        Tree.InsertNode "", 0, NewNode
      End If
    End With
  Next
End Sub

' ------------------------------------------------
' Эта подпрограмма удаляет дочерние Node узлы
' ------------------------------------------------
Sub DeleteChildNodes(Node)
  ' Удалить все дочерние Node узлы
  With Tree
    ' Пока есть дочерние, удалять их
    While .NodeHasChildren(Node)
      .DeleteNode(Tree.ChildNode(Node))
    Wend
  End With
End Sub

' ------------------------------------------------
' Обработчик OnLoad для диалога
' ------------------------------------------------
Sub Dialog_OnLoad
  Dim Root ' корень дерева
  With Tree
    ' Установить системный список картинок
```

```
    ' Свойство Image возвращает номер в этом списке
    .SetSysImageList()
    ' Вставим корневой узел
    Root = .InsertNode("Корреспонденты",ROOT_IMAGE)
    ' Добвим псевдоузел(считаем, что есть дочерние)
    .InsertNode "", 0, Root
    ' Раскрыть узел (вызовется Tree_Expand)
    .NodeExpanded(Root) = True
  End With
End Sub

' ------------------------------------------------
' Обработчик OnExpand для дерева
' ------------------------------------------------
Sub Tree_OnExpand(Node)
  ' Удалим временный узел
  DeleteChildNodes Node
  ' Заполним текущий уровень
  FillTree Tree.NodeParam(Node), Node
End Sub

' ------------------------------------------------
' Обработчик OnCollapse для дерева
' ------------------------------------------------
Sub Tree_OnCollapse(Node)
  ' Удалим все дочерние узлы
  DeleteChildNodes Node
  ' Добавим фиктивный узел,
  ' поскольку были дочерние узлы
  Tree.InsertNode "", 0 , Node
End Sub
```

Объект Workarea

Объект `Workarea` (рабочая область) можно назвать главным объектом программы. Он представляет собой текущую открытую базу данных и, поэтому, содержит в себе практически все остальные объекты. В контексте приложения объект `Workarea` всегда доступен посредством глобальной функции `Workarea()`.

Самое большое количество свойств объекта Workarea предназначено для прямого доступа к объектам и их коллекциям. Имея идентификатор объекта, всегда можно получить на него ссылку, воспользовавшись соответствующим свойством Workarea. Для объектов, представленных коллекциями, имеются свойства, возвращающие нужную коллекцию. Причем, если коллекция многоуровневая (например счета – Accoutnts), то свойство возвращает коллекцию самого верхнего уровня.

Имеются следующие свойства для доступа к многоуровневым объектам и коллекциям.

- **Account** и **Accounts** – доступ к счетам.
- **Agent** и **Agents** – доступ к корреспондентам.
- **Folder** и **Folders** – доступ к папкам документов.
- **Entity** и **Entities** – доступ к объектам учета.
- **Misc** и **Miscs** – доступ к произвольной аналитике.
- **Binder** и **Binders** – доступ к подшивкам.
- **Template** и **Templates** – доступ к шаблонам.

Все эти свойства работают одинаково. Например, чтобы получить ссылку на бухгалтерский счет (объект Account), если известен его идентификатор (ID), нужно обратиться к свойству Account, указав ID в качестве параметра.

Пример

```
Sub DoAccount(ID)
   Dim acc
   Set acc = Workarea.Account(ID)
   ' что-то делаем со счетом
End Sub
```

Если нужно обратиться к коллекции счетов верхнего уровня, то нужно использовать свойство Accounts без параметров.

Пример

```
Sub DoAccounts()
   Dim coll
   Dim acc
   Set coll = Workarea.Accounts
   For Each acc In coll
```

```
        ' что-то делаем со счетом из коллекции
    Next
End Sub
```

Дополнительные свойства объекта `Workarea` обеспечивают доступ к другим сущностям системы, которые либо существуют в единственном экземпляре, либо представлены плоскими (не многоуровневыми) коллекциями.

Это следующие свойства.

- **`Bank, Banks`** – доступ к объектам `Bank`, описывающим банки и коллекции `Banks`.
- **`Cur, Curs`** – доступ к объекту Currency (валюта) и коллекции `Currencies`.
- **`RateKind, RateKinds`** – предоставляют доступ к объекту `RateKind` (вид курса валюты) и коллекции `RateKinds`.
- **`Unit, Units`** – предоставляют доступ к объекту `Unit` (единица измерения) и коллекции `Units`.
- **`PriceList, PriceLists`** – предоставляют доступ к объекту `PriceList` (прайс-лист) и коллекции `PriceLists`.
- **`PriceKind, PriceKinds`** – предоставляют доступ к объекту `PriceKind` (вид цены) и коллекции `PriceKinds`.
- **`ProjectItem, ProjectItems`** – предоставляют доступ к объекту `ProjectItem` (элемент проекта, такой, например, как форма или диалог) и коллекции `ProjectItems`.
- **`TaxReport`** и **`TaxReports`** – доступ к налоговым отчетам.

Для доступа к операциям используются следующие два свойства:

- **`Operation`** – возвращает ссылку на существующую бухгалтерскую операцию (объект `Operation`) по ее `ID`.
- **`Operations`** – возвращает коллекцию операций, построенную по заданному критерию.

Прочие свойства объекта `Workarea`:

- **`DBUser`** – возвращает имя текущего пользователя базы данных. Это имя, под которым пользователь открыл базу данных *Акцент*.

- **UserGroups** – возвращает массив наименований групп, в которые входит текущий пользователь базы данных.
- **AdoConnection** – возвращает ссылку на открытое соединение (объект `ADODB.Connection`) с базой данных. Используется для прямого обращения к базам данных в *SQL* и *MSDE* версиях *Акцент*.
- **DaoDatabase**, **DaoEngine** – эти свойства возвращают ссылки на соответствующие системные объекты ядра баз данных *Microsoft Jet* (только настольные версии *Акцент*)
- **IsDatabaseOpen** – возвращает `True`, если имеется открытая база данных.
- **Assistant** – возвращает ссылку на Ассистента (представлено объектом `DHTMLAssistant`), который связан с рабочей областью (показывается в основном окне приложения).

Методы объекта `Workarea` предназначены для выполнения определенных операций с базой данных. Основные из них следующие.

- **BeginTrans, CommitTrans** и **Rollback** – управление транзакциями.
- **CreateOperation** – создает новую бухгалтерскую операцию, представленную объектом `Operation`.
- **Browse** – выводит диалог выбора объекта базы данных, такого как корреспондент, объект учета и т.д. Метод имеет множество аргументов, задающих поведение диалога выбора. Подробности описаны в электронной справочной системе.
- **CreateReport** – создает объект отчета. Метод получает имя отчета. Работа с отчетами будет описана ниже в этой главе.
- **GetAccCode** и **GetAccID** – позволяют быстро преобразовать код счета в его идентификатор (`ID`) и наоборот.
- **Like** – осуществляет поиск элемента по заданному критерию и возвращает коллекцию найденных элементов.
- **MoveTo** – перемещает один элемент в другую папку или группу.
- **MoveDocTo** – перемещает операцию (документ) в другую папку.

Объект `Workarea` поддерживает набор событий, которые возникают при выполнении определенных операций с объектами базы данных. Эти собы-

тия будут возникать независимо от способа изменения данных, будь то действия пользователя или выполнение программного кода.

Обработчики этих событий должны находиться в специальном модуле, связанном с базой данных.

Фактически, этот модуль начинает выполняться при загрузке базы данных и прекращает свое выполнение только при ее закрытии.

Чтобы связать модуль с базой данных, его нужно специальным образом пометить. Это делается в окне проекта текущей базы данных.

Основные события, которые поддерживаются объектом `Workarea`.

- **`OnLoad`** – возникает после загрузки базы данных, но до появления на экране пользовательского интерфейса.
- **`AfterCreate`** – возникает после создания элемента в базе данных. Тип элемента и его `ID` передается в обработчик события.
- **`AfterDelete`** – возникает после удаления элемента из базы данных. Тип элемента и его `ID` передается в обработчик события.
- **`CanDelete`** – возникает перед удалением элемента из базы данных. Тип элемента и его `ID` передается в обработчик события. Программа имеет возможность отменить удаление объекта.
- **`CanToTrash`** – возникает при попытке удаления операции (документа) в корзину. Программа имеет возможность отменить удаление.
- **`OnChanged`** – возникает после изменения каких-либо свойств объекта в базе данных.
- **`OnProperties`** – возникает при попытке просмотра свойств объекта, причем появление стандартного окна свойств можно заблокировать. Это событие позволяет программисту подставлять вместо стандартных собственные окна свойств.
- **`OnProperties2`** – аналогично событию `OnProperties`, но в обработчик передается дополнительный идентификатор, позволяющий точно идентифицировать объект, вызвавший срабатывание события.
- **`OnAllTasksMenu`** – это событие возникает перед выводом на экран подменю *«Все задачи»* контекстного меню элемента. Обработчик может дополнить меню своими пунктами. При выборе пользовательского пункта меню возникает событие `OnPopup`.

- **OnToBinder** – возникает при помещении операции или документа в подшивку (Binder).
- **OnToFolder** – возникает при помещении операции или документа в другую папку (Folder).
- **OnReportStart** – возникает перед запуском любого отчета, как стандартного, так и созданного пользователем. Перехват этого события позволяет блокировать запуск отчетов или подставлять вместо стандартных отчетов другие, созданные программистом.

Объект Workarea поддерживает работу с пользовательским таймером, который позволяет организовать дополнительную обработку данных в заданные промежутки времени. Таймер устанавливается с помощью метода SetTimer. После того, как таймер установлен, периодически возникает событие OnTimer.

Объекты и база данных

Многие объекты *Акцент* представляют собой сущности, которые сохраняются в базе данных. Примерами таких объектов могут служить счет (Account), корреспондент (Agent), объект учета (Entity) и многие другие. Все такие объекты имеют несколько общих особенностей.

Все объекты, которые могут храниться в базе данных, имеют уникальный идентификатор. Идентификатор представляет собой целое значение, которое гарантированно не повторяется в пределах текущей базы данных. Это значение используется внутри базы данных в качестве *первичного ключа*. Идентификатор объекта возвращается свойством только для чтения ID. Если ID объекта равен нулю, то объект еще не сохранен в базе данных. Идентификаторы используются для связи объектов между собой.

Вторая особенность хранимых объектов заключается в способе записи измененных значений в базу данных. При любых изменениях свойств объектов немедленная запись в базу данных не выполняется. Вместо этого, измененное значение свойства запоминается и устанавливается специальный флажок Modified. Запись изменений в базу данных производится вызовом метода Save. При успешной записи изменений, флажок Modified сбрасывается (устанавливается в False). Такое поведение объектов позволяет значительно повысить быстродействие приложения при установке большого количества свойств.

Объекты Operation, Transaction, TransList

Эти три объекта по праву можно считать основой программы. Объект `Operation` представляет собой бухгалтерскую операцию. Операция может состоять из одного или нескольких списков проводок. Список проводок представлен объектом `TransList`. Список, в свою очередь, содержит одну или несколько бухгалтерских проводок. Проводка представлена объектом `Transaction`. На рисунке ниже приведена связь этих объектов между собой.

Ошибка! Ошибка связи.

Рис. 1. Объект Operation

Кратко опишем то, что изображено на рисунке. Внутри объекта Operation находится единственный объект `TransList`. Кажется, что он представляет собой одну проводку, видимую в журнале операций.

Однако, поскольку в журнале проводка может быть многострочной, и все свойства каждой из строк, включая счета, корреспондентов, суммы, количества и даже даты могут устанавливаться независимо, то реально проводкой (`Transaction`) является одна строка проводки из журнала операций. Именно это и показано на рисунке.

Для более удобного использования этих объектов в простых случаях, когда списки проводок содержат по одной строке, в объект `Operation` было добавлено специальное свойство `Trans`, которое позволяет получить ссылку на объект `Transaction`, минуя промежуточный объект `TransList`. У этого свойства два аргумента. Первый задает номер проводки, а второй – номер строки в проводке. Причем, второй аргумент является опциональным, и если он опущен, свойство возвращает первую строку заданной проводки, что и требовалось.

Имеется также свойство `TransList`, позволяющее получить объект, представляющий собой сам список проводок.

Объект `TransList` чаще всего используется в случаях, когда нужно оперировать со всеми строками проводки как с единым целым. У объекта `TransList` имеется свойство Item, возвращающее ссылку на одну строку проводки.

Таким образом следующие записи полностью эквивалентны:

```
op.Trans(1)
op.Trans(1,1)
```

```
op.TransList(1).Item(1)
```

Все они возвращают ссылку на первую строку первой проводки.

Объект Operation

Объект `Operation` представляет собой бухгалтерскую операцию. Он может быть получен с помощью свойства `Operation` объекта `Workarea` как элемент коллекции операций `Operations` или как текущая операция, связанная с формой или диалогом (свойство `CurrentOperation`).

Основные свойства объекта `Operation`:

- **`Date`** – дата операции. Установка даты операции автоматически устанавливает даты для всех входящих в нее проводок. Такое поведение можно изменить, установив в `False` значение свойства `LockDate`.
- **`DocNo`** –номер документа.
- **`Done`** – признак проведенности операции.
- **`ID`** – идентификатор операции. Пока операция новая, то есть ни разу не сохранялась в базе данных, значение свойства равно нулю.
- **`Folder`** – папка, в которой находится операция. Свойство возвращает ссылку на объект `Folder`. Свойство только для чтения. Для того чтобы установить папку, в которую нужно поместить операцию, можно воспользоваться свойством `FolderID`.
- **`FolderID`** – идентификатор папки, в которой хранится операция.
- **`IsNew`** – равно `True`, если операция еще ни разу не сохранялась. Для такой операции значение идентификатора (`ID`) равно нулю.
- **`FormID`** – идентификатор формы, с которой связана данная операция. Сами формы представлены объектом `ProjectItem`.
- **`FormName`** – краткое наименование формы, с которой связана данная операция. Если операция не связана с формой, возвращается пустая строка. Свойство только для чтения.
- **`AutoNumID`** – идентификатор элемента автонумерации. Значение свойства автоматически устанавливается при связывании операции с формой. Если свойство не равно нулю, то при сохранении операции, ей автоматически присваивается номер документа (`DocNo`).
- **`Name`** – наименование операции. Длина наименования не более 255 символов.

- **Name2** – наименование операции. Отличается от свойства Name тем, что устанавливается только тогда, когда наименование пусто. Обычно используется в обработчиках шаблонов.
- **Memo** – примечание к операции. Длина примечания не может быть больше 255 символов.
- **Memo2** – примечание к операции. Отличается от свойства Memo тем, что устанавливается только тогда, когда примечание пусто. Обычно используется в обработчиках шаблонов.
- **Tag** – произвольный текстовый признак операции. Длина не более 50 символов.
- **Params** – коллекция параметров операции. Свойство только для чтения и возвращает ссылку на коллекцию Params.
- **Modified** – если равно True, то операция изменена.
- **Sum** – сумма операции (справочно). Если операция состоит из одной проводки, то сумма операции автоматически будет равна сумме первой и единственной проводки.
- **TransCount** – количество проводок.
- **TemplateID** – ID шаблона, с помощью которого создана операция. Установка свойства не приводит к выполнению шаблона. Для того чтобы выполнить шаблон используется специальный объект TemplateInvoker (исполнитель шаблона). Ссылку на этот объект можно получить с помощью одноименного свойства TemplateInvoker.
- **TransList** – объектное свойство возвращает список проводок (объект TransList).
- **Trans** – объектное свойство возвращает одну строку проводки (объект Transaction). Свойство имеет два аргумента – номер проводки и номер строки в проводке. Второй аргумент (номер строки) необязательный. Это позволяет более просто записывать ссылки в случае однострочных проводок.

Объект Operation имеет несколько дополнительных атрибутов, которые можно использовать произвольно. Эти атрибуты имеют разные типы и доступны с помощью следующих свойств.

- **String1**, **String2**, **String3** – три атрибута типа String.
- **Date1**, **Date2**, **Date3** – три атрибута типа Date.
- **Sum1**, **Sum2**, **Sum3** – три атрибута типа Currency.
- **Long1**, **Long2**, **Long3** – три атрибута типа Long.

Операции могут быть связаны друг с другом в отношении один ко многим. Это означает, что каждая операция может иметь только одну родительскую операцию и сколько угодно дочерних операций. Для связывания операций используются следующие свойства и методы.

- **Link** – свойство задает идентификатор связанной (родительской) операции. Установка свойства не приводит к немедленной записи изменений в базу данных, а просто взводит флажок `Modified`. Запись осуществляется в момент вызова метода `Save`.
- **Linked** – объектное свойство только для чтения. Возвращает коллекцию связанных (дочерних) операций. Операция связана с текущей, если значение свойства `Link` равно идентификатору текущей операции. Возвращаемая коллекция относится к типу `Operations`.
- **LinkTo** – метод, позволяющий связать текущую операцию с указанной. В отличие от свойств, метод выполняется немедленно.

Объект `Operation` имеет несколько методов, позволяющих копировать информацию из одной проводки в другую. Эти методы чаще всего используются в обработчиках шаблонов, хотя никто не мешает использовать их когда нужно.

- **CopyAccounts** – копирует счета из одной проводки в другую.
- **CopyAgents** – копирует корреспондентов из одной проводки в другую.
- **CopyEntities** – копирует объекты учета из одной проводки в другую.
- **CopyMiscs** – копирует произвольную аналитику из одной проводки в другую.
- **CopyCurs** – копирует валюту.
- **CopySeries** – копирует серии (партии) объектов учета.

Объект имеет несколько вспомогательных методов и свойств, облегчающих программирование.

- **Binders** – объектное свойство только для чтения. Возвращает коллекцию подшивок (объект `Binders`), в которые входит данная операция.

- **LockRecalc** – свойство позволяет изменить алгоритм пересчета свойств «цена – количество - сумма» в проводках. Если свойство равно True, то никакой автоматический пересчет не выполняется и ответственность за правильность данных полностью ложится на программиста.
- **Clone** – создает копию текущей операции. Обратите внимание, что создается не еще одна ссылка, а именно реальная копия операции. Метод имеет несколько флажков, позволяющих включать/выключать копирование некоторых дополнительных атрибутов, таких как произвольная аналитика, валюта и параметры проводок.
- **Autonum** – метод вызывает принудительное присваивание номера документа. Обратите внимание, что для исключения дублирования номеров документов метод «одноразовый», то есть при следующем вызове он вернет следующий по порядку номер.
- **Pack** – метод удаляет «лишние» пустые строки в проводках. Строка считается пустой, если для нее не указан объект учета.
- **SetDate** – метод устанавливает дату операции и дату всех входящих в нее проводок.
- **SetFolder** – метод устанавливает идентификатор папки, в которую нужно поместить операцию. Если метод вызывается для уже сохраненной операции, то при сохранении операции она помещается в указанную папку.
- **New** – делает операцию «новой», то есть сбрасывает в нуль все идентификаторы проводок, параметров и т.д.
- **SetOwner** – позволяет установить владельца операции. Владельцем считается корреспондент (типа «Моя фирма»), от имени которого ведется учет. Обратите внимание, что владельца можно установить и для «чужой» фирмы. Такое поведение позволяет выполнять «зеркальные» операции для нескольких предприятий одновременно.
- **EnumTrans** – вызывает функцию обратного вызова для каждой из проводок операции. Ссылка на функцию обратного вызова должна быть получена с помощью функции `GetRef`.
- **FindTrans** – выполняет поиск заданной проводки. Для поиска метод вызывает функцию обратного вызова, которая должна вернуть `True`, если проводка соответствует заданному критерию.

Для работы с пользовательским интерфейсом объект `Operation` имеет несколько методов, вызывающих диалоговые окна.

- **`ShowTrans`** – показывает стандартное диалоговое окно проводок операции.
- **`ShowBinders`** – показывает окно со списком подшивок, в которые входит операция.
- **`ShowLinks`** – показывает стандартное диалоговое окно связанных документов.

Для более удобной работы в формах и диалогах, объект имеет специальный метод `CanClose`.

- **`CanClose`** – проверяет, была ли изменена операция и предлагает сохранить ее. Для выполнения дополнительных действий в процессе сохранения операции, в метод можно передать *ссылку на пользовательскую функцию*.

Для блокировки одновременной работы пользователей сети с одной операцией имеется пара методов `Lock/Unlock` (только SQL-версия).

- **`Lock`** – блокирует операцию до вызова метода `Unlock` или до завершения сеанса работы с сервером (в том числе и аварийного).
- **`Unlock`** – снимает блокировку, наложенную методом `Lock`.

Существует возможность сохранить вместе с операцией произвольную информацию большого объема (до 2GB). Для этого используется парам методов:

- **`SetExtraData`** – записывает дополнительную информацию, связанную с операцией.
- **`GetExtraData`** – возвращает информацию, записанную методом `SetExtraData`.

Объект TransList

Объект `TransList` представляет собой список проводок. Каждая проводка представлена объектом `Transaction`. Основные свойства объекта `TransList` следующие.

- **`Sum`** – возвращает общую сумму для всех строк списка. Свойство только для чтения.

- **CurSum** – свойство только для чтения, которое возвращает сумму в заданной валюте для всех строк из списка. Порядковый номер валюты задается в качестве аргумента. Обратите внимание, что это именно порядковый номер, а не идентификатор валюты.
- **Qty, FQty** – возвращают общее количество в базовой или производной единицах измерения. Свойства только для чтения. Обратите внимание, что использовать данные свойства имеет смысл только тогда, когда все объекты учета в списке одинаковые.
- **Rows** – возвращает количество строк в списке проводок.
- **Item** – свойство только для чтения, которое возвращает ссылку на объект `Transaction`, который представляет собой одну строку из списка. Свойство имеет целый аргумент, задающий номер строки (от 1 до `Rows`).

Практически все методы объекта `TransList` предназначены для работы со всеми проводками списка как с единым целым. Эти методы таковы:

- **SetDate** – устанавливает дату для всех строк списка.
- **SetAccDb, SetAccCr** – устанавливают счета по дебету или по кредиту для всех строк списка.
- **SetAgFrom, SetAgTo** – устанавливают корреспондентов «Кто» и «Кому» для всех строк списка.
- **SetAgMcFrom, SetAgMcTo** – устанавливает корреспондентов «Кто» и «Кому» для всех строк списка равным текущей выбранной фирме, от имени которой ведется учет.
- **SetEnt** – устанавливает объект учета для всех строк списка.
- **SetMisc** – устанавливает аналитический показатель для всех строк списка. Свойство имеет дополнительный аргумент, задающий номер аналитики. Обратите внимание, что это номер аналитики, а не порядковый номер (его можно посмотреть в окне свойств типа аналитики).
- **SetRate** – устанавливает курс валюты для всех строк списка. Свойство имеет дополнительный аргумент, задающий порядковый номер валюты. Установка курса приводит к автоматическому пересчету сумм.

- **SetCur** – устанавливает идентификатор валюты для всех проводок списка.
- **SetTax** – устанавливает указанный адрес в налоговых отчетах для всех проводок списка. Можно одновременно задавать несколько адресов, разделенных точкой с запятой.

Помимо методов, служащих для установки свойств проводок, имеются дополнительные методы и свойства, связанные с обработкой шаблонов. Основные из них следующие.

- **CalcSum** – (метод) вычисляет суммы по проводкам по заданной формуле.
- **CalcQty** – (метод) вычисляет количества по проводкам по заданной формуле.
- **LockUI** – (свойство) устанавливает признак блокировки ввода для элементов интерфейса пользователя в журнале операций. Более подробно свойство описано в электронной документации.

Эти методы можно использовать только в обработчиках шаблонов. Попытка использования в другом месте приведет к ошибке выполнения программы. Более подробно эти методы описаны в разделе, посвященном обработчикам шаблонов (стр. 123).

Вспомогательные методы объекта `TransList` следующие.

- **Clear** – удаляет все строки, кроме первой и очищает первую строку проводки.
- **Pack** – удаляет из списка все проводки, для которых не указан объект учета.
- **InsertRow,DeleteRow** – позволяют вставлять или удалять строки в проводку.
- **SetLockRate** – устанавливает свойство `LockRate` для всех строк проводки. Если значение свойства равно `True`, то при пересчетах сумм курс меняться не будет.
- **Sort** – позволяет выполнить сортировку строк по заданному критерию. Можно сортировать по дате, сумме, цене и количеству.
- **SwapRows** – переставляет местами две строки в списке проводок. Используется для организации специальных сортировок совместно с глобальной функцией `QuickSort`.

- **EnumTrans** – вызывает функцию обратного вызова для каждой из проводок списка. Ссылка на функцию обратного вызова должна быть получена с помощью функции GetRef.
- **FindTrans** – выполняет поиск заданной проводки в списке. Для поиска метод вызывает функцию обратного вызова, которая должна вернуть True, если проводка соответствует заданному критерию.

При программировании различного поведения проводок в списке часто бывает нужно управлять «наследованием» различных аналитических показателей при добавлении строк в проводку. Для этого используется специальное свойство Inherit. Значение свойства представляет собой набор флажков, устанавливая и сбрасывая которые можно включать или выключать наследование различных аналитических показателей при добавлении строк в список. Более подробная информация об использовании этого свойства приведена в электронной справочной системе.

Объект Transaction

Объект Transaction представляет собой одну строку проводки. Проводки группируются в списки (объект TransList). Списки, в свою очередь, хранятся в объекте Operation. Ссылка на проводку может быть получена из списка проводок (TransList) с помощью свойства Item либо непосредственно из операции (Operation) с помощью свойства Trans.

Основные свойства позволяют устанавливать суммы, даты и аналитические показатели для проводки.

- **Date** – дата проводки. Свойство может устанавливаться автоматически при установке даты в родительском списке проводок или в родительской операции.
- **Sum** – сумма проводки.
- **Qty**, **FQty** – количество для проводки в базовой и производной единицах измерения.
- **Price** – цена объекта учета в базовой единице измерения.
- **AccDbID**, **AccCrID** – свойства, которые возвращают или устанавливают идентификаторы счетов по дебету и кредиту для проводки.

- **AccDb**, **AccCr** – свойства только для чтения, которые возвращают ссылки на счета (объект `Account`) по дебету и кредиту для проводки.
- **AgFromID**, **AgToID** – свойства, которые возвращают или устанавливают идентификаторы корреспондентов «Кто» и «Кому» для проводки.
- **AgFrom**, **AgTo** – объектные свойства только для чтения, которые возвращают ссылки на корреспондентов (объект `Agent`) «Кто» и «Кому» для проводки.
- **EntID** – свойство возвращает или устанавливает идентификатор объекта учета для проводки.
- **Entity** – объектное свойство только для чтения, которое возвращает ссылку на объект учета (объект `Entity`) для проводки.
- **UnitID**, **FUnitID** – возвращают или устанавливают идентификаторы базовой и производной единиц измерения для объекта учета проводки.
- **Unit**, **FUnit** – объектные свойства только для чтения, возвращающие ссылки на единицу измерения (объект `Unit`) для базовой и производной единиц измерения.
- **MiscID** – возвращает или устанавливает идентификатор произвольной аналитики для проводки. Свойство имеет один аргумент, задающий номер (не порядковый!) аналитики.
- **Misc** – объектное свойство только для чтения возвращает ссылку на произвольную аналитику (объект `Misc`). Как и для свойства `MiscID` нужно задать номер возвращаемой аналитики.
- **SeriesID** – возвращает или устанавливает идентификатор партии (серии) объекта учета.
- **Series** – объектное свойство только для чтения, возвращающее ссылку на партию (серию) объекта учета (объект `Series`) для проводки.
- **TransCurs** – объектное свойство, возвращающее коллекцию валют (объект `TransCurrencies`) , связанных с данной проводкой. Работа с валютой описана ниже.

Объект Transaction имеет некоторое количество вспомогательных свойств для облегчения программирования.

- **IsEmpty** – свойство только для чтения возвращает True если проводка пуста, то есть для нее не установлен объект учета.
- **ID** – свойство возвращает идентификатор проводки в базе данных. Используется для связывания проводок между собой.
- **RowNo**, **RowNo2** – свойства возвращают номер проводки в списке. Обычно используются для нумерации строк в документах.
- **Operation** – свойство только для чтения возвращает ссылку на родительскую операцию, которая содержит данную проводку.

Проводки и налоговые отчеты

У объекта Transaction есть свойство, которое связывает конкретную проводку с налоговыми отчетами.

- **TransTax** – возвращает ссылку на объект TransTax, который предназначен для связывания проводки с налоговым отчетом.

Объект TransTax имеет, в свою очередь, свойство, задающее адрес (метку) внутри налогового отчета.

- **Addr1** – метка внутри налогового отчета.

Связывание свойств с элементами управления

Объект Transaction имеет большую группу свойств, которые возвращают специальные объекты, предназначенные для связывания свойств с элементами управления.

Связывание означает, что элемент управления на форме или в диалоге будет «отслеживать» изменение значения связанного свойства и наоборот, изменения текста в элементе управления будут приводить к изменению значения свойства. Такой подход позволяет значительно сократить размер программы и упростить ее логику.

Обратите внимание, что объекты для связывания не содержат внутри себя никаких данных. Изменение свойств объектов для связывания фактически приводит к изменению свойств того объекта, который вернул связанный объект.

Приведем простой пример. Пусть на форме имеется селектор (редактор с кнопкой), который должен выводить объект учета из первой строки первой проводки бухгалтерской операции.

Мы не можем написать в источнике данных `DataSource` селектора что-то вроде

```
op.Trans(1,1).Entity.Name
```

Так написать нельзя из-за того, что если в проводке не указан объект учета, то свойство `Entity` вернет `Nothing` и попытка получить значение свойства `Name` приведет к ошибке выполнения.

Для решения этой проблемы и существует объект `EntityBind`, связанный с проводкой. Ссылку на него возвращает свойство `EntBind` объекта `Transaction`.

Свойства объекта `EntityBind` возвращают значения независимо от того, существует родительский объект (в нашем случае `Entity`) или нет. Если объект не существует (равен `Nothing`), то свойство просто возвращает пустую строку.

Таким образом правильная запись будет такой:

```
op.Trans(1,1).EntBind.Name
```

При попытке записи какого-либо свойства объект ведет себя более интересным образом. Записываемый текст рассматривается как фрагмент значения связанного свойства и по нему осуществляется поиск. Если объект учета найден, то он устанавливается в качестве родительского (в нашем случае устанавливается значение свойства `EntID` родительского объекта `Transaction`).

При связывании свойств с элементом управления типа селектора (редактор с кнопкой) имеется еще одна дополнительная возможность. При нажатии кнопки селектора система выводит диалог выбора соответствующего объекта.

Обратите внимание. Объекты для связывания очень тесно связаны с родительскими объектами и не могут существовать отдельно от них. Не пытайтесь сохранять ссылки на объекты для связывания во временных переменных. Это может привести к нарушению связей и неправильной работе таких объектов.

Проводка имеет следующие свойства, возвращающие объекты для связывания.

- **AccDbBind**, **AccCrBind** – связанные счета по дебету и кредиту.
- **AgToBind**, **AgFromBind** – связанные корреспонденты «Кому» и «Кто».
- **EntBind** – связанный объект учета.
- **MiscBind** – связанная аналитика. Свойство имеет аргумент, задающий номер (не порядковый!) произвольной аналитики.
- **UnitBind**, **FunitBind** – связанные базовая и производная единицы измерения для объекта учета.
- **SeriesBind** – связанная партия (серия) для объекта учета проводки.

Более подробная информация об объектах, предназначенных для связывания, приведена в электронной справочной системе.

Работа с валютой

Каждая проводка в программе, помимо суммы в базовой учетной валюте, может иметь произвольное количество сумм в разных валютах. Причем, при построении отчетов валютные суммы полностью самостоятельны и никак не связаны с суммами в основной валюте. Допустимо проводить несколько сумм в одной и той же валюте но по разным курсам.

Для работы с валютами в проводках используется специальный объект `TransCurrency`. Объект хранит информацию об сумме в одной валюте.

Свойства объекта `TransCurrency` следующие:

- **CurID** – идентификатор валюты.
- **Cur** – объектное свойство только для чтения, которое возвращает ссылку на валюту (объект `Currency`).
- **CurBind** – объектное свойство, которое возвращает ссылку на специальный объект `CurrencyBind`, служащий для связывания свойств валюты с элементами управления.
- **Sum** – сумма в валюте.
- **Price** – цена в валюте.

- **Rate** – курс валюты. Обратите внимание, что курс устанавливается в соответствии со значением делителя валюты Denom. Например, если для USD установлен Denom равный 100, то курс 5.42:1 нужно устанавливать как 542 (542:100).
- **No** – свойство возвращает порядковый номер элемента в списке валют.

Объекты TransCurrency хранятся в коллекции TransCurrencies, которая, в свою очередь, хранится внутри проводки Transaction и ссылка на нее возвращается свойством TransCurs. Основные свойства коллекции TransCurrencies такие же, как и у других коллекций.

- **Count** – возвращает количество элементов в коллекции
- **Item** – возвращает элемент коллекции по индексу.
- **IsEmpty** – возвращает True, если коллекция пуста (Count = 0).

Объект Account и коллекция Accounts

Объект Account представляет собой бухгалтерский счет. Счета в программе организованы в многоуровневую коллекцию Accounts. Как и все объекты, организованные в многоуровневую коллекцию, объект Account поддерживает свойства Children, HasChildren, Type, Image, Nested, ID, Name, Tag, Memo.

Дополнительные свойства объекта Account следующие.

- **PlanID** – идентификатор плана счетов, которому принадлежит данный счет.
- **Code** – код счета.
- **SaldoType** – тип свертки сальдо для пассивно-активного счета.

Коллекция Accounts имеет только одну особенность. Верхний уровень счетов может содержать только планы счетов. Это означает, что в коллекции верхнего уровня Accounts, которая получена с помощью свойства Accounts объекта Workarea, можно создавать счета только с типом –1. (–1 означает план счетов).

В остальном коллекция полностью аналогична всем остальным многоуровневым коллекциям.

Объект Agent и коллекция Agents

Объект `Agent` представляет собой корреспондента. Объекты этого типа организованы в многоуровневую коллекцию. Корреспонденты могут находиться в коллекции `Agents`.

Основные свойства объекта `Agent` следующие.

- **`Name`** – наименование корреспондента.
- **`Code`** – код для корреспондента. Для предприятий это код ОКПО (ЕДРПОУ).
- **`Tag`** – произвольная текстовая строка, связанная с корреспондентом.
- **`Memo`** – примечание к корреспонденту.
- **`VatNo`** – код плательщика налогов. Для предприятий это код плательщика НДС, для физических лиц и сотрудников предприятий – индивидуальный налоговый номер (ИНН).
- **`RegNo`** – номер свидетельства о регистрации плательщика НДС.
- **`TabNo`** – табельный номер для корреспондентов типа сотрудник или физическое лицо.
- **`Phone`** – номер телефона корреспондента.
- **`Passport`** – паспортные данные корреспондента.
- **`EMail`**, **`WWW`** – электронный и WWW адреса корреспондента.
- **`Contact`** – имя контактного лица.
- **`Country`** – страна, в которой находится корреспондент.
- **`DateIn, DateOut`** – даты приема и увольнения для корреспондентов типа сотрудник.

Обратите внимание. Некоторые свойства корреспондентов различных типов недоступны из пользовательского интерфейса программы. Однако, это не делает их недоступными из кода на *VBScript*. В этом случае их можно использовать произвольно.

Коллекция корреспондентов `Agents` каких-либо особенностей не имеет.

Объект Entity и коллекция Entities

Объект `Entity` представляет собой объект учета. Все типы объектов обладают одинаковым набором свойств, хотя использование их зависит от типа объекта учета. Например, для объектов денежного типа невозможно (из интерфейса пользователя) установить свойства `Cat` и `Nom`, хотя использовать их из программы не запрещается.

Основные свойства объекта `Entity` следующие.

- **`Name`** – наименование объекта учета.
- **`Cat`** – каталожный номер объекта учета.
- **`Nom`** – номенклатурный номер объекта учета.
- **`Art`** – артикул объекта учета.
- **`Bar`** – бар-код для объекта учета.
- **`Tag`** – произвольная текстовая строка, связанная с объектом учета.
- **`Memo`** – примечание к объекту учета.
- **`UnitID`** – идентификатор единицы измерения, связанной с объектом учета. Эта единица измерения обычно автоматически подставляется в проводку при выборе этого объекта учета.

Коллекция объектов учета `Entities` каких-либо особенностей не имеет.

Объект EntUnit и коллекция EntUnits

Объекты учета поддерживают набор производных единиц измерения. Каждая производная единица фактически представляет собой ссылку на еще одну единицу измерения и множитель/делитель для пересчета одной в другую.

Весь этот набор свойств представлен объектом `EntUnit`.

- **`UnitID`** – идентификатор производной единицы измерения.
- **`Num`/`Denom`** – множитель и делитель для пересчета производной единицы измерения в базовую.
- **`UnitName`**, **`UnitShortName`** – возвращают наименование и сокращение для единицы измерения, установленной свойством `UnitID`. Если единица измерения не установлена – возвращается пустая строка. Свойства предназначены для упрощения связывания объекта с элементами управления на формах и в диалогах.

Объекты EntUnit находятся в коллекции EntUnits. Получить ссылку на эту коллекцию можно с помощью свойства EntUnits объекта учета Entity.

Объект Binder и коллекция Binders

Объект Binder представляет собой подшивку документов (операций). Помимо общих свойств и методов для многоуровневых объектов он имеет следующие свойства и методы.

- **Operations** – объектное свойство только для чтения, которое возвращает коллекцию операций, входящих в подшивку.
- **AddOperation** – метод добавляет операцию в подшивку.
- **RemoveOperation** – метод удаляет операцию из подшивки. Обратите внимание, что метод не удаляет операцию, а просто разрывает ее связь с подшивкой.

Коллекция подшивок Binders имеет следующую особенность. Существует возможность получить коллекцию Binders подшивок, в которые входит заданная операция. Такая коллекция возвращается свойством Binders объекта Operation. Такая коллекция будет «плоской», а не многоуровневой. Не рекомендуется использование в такой коллекции методов и свойств, связанных с многоуровневостью.

Объект Template и коллекция Templates

Объект Template представляет собой шаблон типовой бухгалтерской операции. Объект является типичным представителем многоуровневой коллекции и, помимо общих, имеет следующие методы и свойства.

- **FormID** – свойство возвращает или устанавливает идентификатор связанной с шаблоном формы или диалога.
- **Linked** – объектное свойство только для чтения возвращает ссылку на коллекцию связанных с текущим шаблонов (объект Templates).
- **GetScript**, **SetScript** – эти методы позволяют прочитать и записать текст программы шаблона. Обычно используются при переносе данных между различными базами данных.

Коллекция шаблонов Templates каких-либо особенностей не имеет.

Как связаны шаблоны и бухгалтерские операции описано далее в этой главе.

Объект Param и коллекция Params

Объект `Param` представляет собой параметр объекта. Параметры группируются в коллекцию `Params`. Если объект поддерживает параметры, то у него имеется одноименное свойство `Params` (только для чтения), которое и возвращает ссылку на коллекцию `Params`.

Основные свойства объекта `Param`.

- **`ID`** – идентификатор параметра в базе данных. Свойство только для чтения.
- **`Name`** – наименование параметра.
- **`Value`**, **`Value2`** – значения параметра. `Value2` отличается от `Value` тем, что если параметр равен `Null`, то свойство `Value2` вернет подходящее ненулевое значение (например пустую строку для строковых параметров). Более подробная информация приведена в электронной документации.
- **`Value3`** – значение параметра в виде строки с разрешением ссылок. Если параметр не является ссылкой, то поведение свойства полностью повторяет `Value2`. При чтении параметра – ссылки делается попытка разрешения ссылки и возвращается нужное свойство объекта на которое ссылается параметр (на какое именно свойство ссылается параметр определяется аргументами при вызове свойства). При записи свойства-ссылки выполняется поиск нужного объекта по содержимому текстовой строки и параметру присваивается идентификатор найденного (и выбранного) объекта.
- **`ValueString`** – значение параметра в виде строки. Обычно используется для связывания с элементами управления.
- **`IsNull`** – свойство возвращает `True`, если значение параметра (свойство `Value`) равно `Null`.

Параметры целого типа (`Long`) могут представлять собой ссылки. Для работы со ссылками имеются дополнительные свойства.

- **`RefKind`** – тип (вид) параметра-ссылки.
- **`RefID`** – идентификатор для параметра-ссылки.

Параметры хранятся в коллекции Params, которая имеет следующие методы и свойства.

- **Count** – возвращает количество параметров в коллекции
- **Item** – возвращает параметр. Обратите внимание, что в качестве аргумента свойства используется имя параметра, а не его индекс в коллекции.
- **IsEmpty** – возвращает True, если коллекция пуста (Count = 0).
- **Exists** – метод проверяет наличие параметра с заданным именем.
- **Create**, **CreateRef** – эти два метода используются для создания параметров или параметров-ссылок.
- **Remove** – метод удаляет параметр из базы данных. Будьте внимательны, метод удаляет ВСЕ значения заданного параметра.
- **Refresh** – этот метод заново считывает все внутренние кэши коллекции из базы данных.

Объект Fact и коллекция Facts

Объект Fact представляет собой факт. *Фактом* в системе называется параметр, зависящий от времени. Другими словами, значение факта в разные моменты времени может быть различным. Факт характеризуется именем, типом и историей – набором пар «дата – значение». Факты группируются в коллекцию Facts.

В отличие от параметров, коллекция фактов всегда связана с конкретной датой. Поэтому у свойства Facts, которое и возвращает коллекцию Facts имеется дополнительный аргумент, задающий дату, на которую нужно получить значения фактов. Как и параметры, факты типа Long могут представлять собой ссылки на другие объекты системы.

Свойства объекта Fact.

- **ID** – идентификатор факта в базе данных. Свойство только для чтения.
- **Name** – наименование факта.
- **Value**, **Value2** – значения факта. Value2 отличается от Value тем, что если факт равен Null, то свойство Value2 вернет подхо-

дящее ненулевое значение (например пустую строку для строковых фактов).

- **IsNull** – свойство возвращает `True`, если значение факта (свойство `Value`) равно `Null`.
- **ActualDate** – реальная дата, на которую было введено значение факта.
- **RefKind** – тип (вид) факта-ссылки.
- **RefID** – идентификатор для факта-ссылки.
- **History** – возвращает двумерный массив пар «дата-значение» для истории значений факта.

Объект `Fact` имеет несколько методов:

- **SetNull** – устанавливает значение факта в `Null`.
- **BrowseHistory** – показывает диалог истории значений факта.

Факты хранятся в коллекции `Facts`. Эта коллекция имеет следующие основные свойства и методы.

- **Count** – возвращает количество фактов в коллекции
- **Item** – возвращает факт. Обратите внимание, что в качестве аргумента свойства используется имя факта, а не его индекс в коллекции.
- **IsEmpty** – возвращает `True`, если коллекция пуста (`Count` = 0).
- **Exists** – метод проверяет наличие факта с заданным именем.
- **Create** – этот метод используются для создания фактов.
- **Remove** – метод удаляет факт из базы данных. Будьте внимательны, метод удаляет ВСЕ значения заданного факта на все даты.
- **Refresh** – этот метод заново считывает все внутренние кэши коллекции из базы данных.

Объект TaxReport и коллекция Tax Reports

Объект `TaxReport` представляет собой описатель налогового отчета. *Налоговый отчет* это специальный элемент системы, специально предназначенный для заполнения бланочных отчетов.

Любую проводку программы можно «пометить» специальными метками, связанными с каждым из налоговых отчетов. Обычно в качестве такой метки выступает просто адрес (номер строки и номер колонки) ячейки отчета, в которую должна попасть сумма проводки. Таким образом, построение любых отчетов типа деклараций или ведомостей предельно упрощается. Специальный отчет `RepTax` помогает собрать информацию о помеченных проводках.

Свойства объекта `TaxReport`.

- **`FileName`** – имя файла электронной таблицы, которая связана с отчетом.
- **`ID`** – идентификатор налогового отчета. Свойство только для чтения.
- **`Memo`** – примечание к налоговому отчету.
- **`Modified`** – признак того, что свойства отчета изменялись после последнего сохранения. Сохранение изменений выполняется методом **`Save`**.
- **`Name`** – наименование налогового отчета.
- **`Tag`** – произвольный текстовый признак налогового отчета.

Налоговый отчет `TaxReport` имеет единственный метод:

- **`Save`** – сохраняет изменения свойств отчета в базе данных.

Как и многие сущности программы, налоговые отчеты объединены в коллекцию `TaxReports`. Это обычная коллекция без каких либо особенностей. Ссылка на коллекцию `TaxReports` может быть получена из объекта `Workarea` с помощью свойства `TaxReports`.

Работа с шаблонами

Шаблон в программах семейства *Акцент* представлен обычной программой на языке *VBScript*. Основное отличие шаблона от обычного модуля заключается в том, что фактически шаблон состоит из обработчиков нескольких событий. Эти события возникают в процессе применения шаблона к бухгалтерской операции, а также при любых изменениях в операции.

Представление шаблона в виде программы очень удобно для программиста, но неприемлемо для пользователя. Чтобы решить эту проблему в программу встроен специальный разборщик программы.

Текст программы размечается специальными комментариями. По команде **Изменить** система разбирает текст шаблона и представляет его в диалоге в виде, удобном для редактирования. Комментарии начинаются с символов {{ и заканчиваются парой символов }} . Все, что находится внутри этих комментариев, изменять вручную не рекомендуется, так как при внесении изменений в диалоговом режиме все исправления потеряются. Вне этих комментариев можно писать любой код на *VBScript*.

Шаблоны поддерживают следующие события.

- **OnApply** – возникает при первом применении шаблона к операции.
- **OnEdit** – возникает вначале редактирования операции, которая уже связана с шаблоном.
- **OnRecalc** – возникает при любых изменениях в операции.
- **BeforeSave** – возникает перед сохранением операции (фактически перед вызовом метода `Save` объекта `Operation`).
- **AfterSave** – возникает после сохранения операции (после вызова метода `Save`).

Все события имеют один обязательный аргумент, который представляет собой ссылку на редактируемую операцию.

Когда операция редактируется в форме или диалоге, процесс обработки шаблона выглядит несколько сложнее. Сама форма (или диалог) содержит выполняемый код, который тоже может обрабатывать текущую операцию. Попытка совместить выполнение двух программ (из формы и из шаблона) приводит к путанице. Предсказать поведение программы в этом случае достаточно сложно. Но использовать шаблоны в формах очень удобно. Это позволяет создать одну форму и использовать ее для проведения самых разных операций, используя шаблоны как схемы проводок и, возможно, для простого расчета сумм.

Для решения описанной проблемы предназначен специальный объект `TemplateInvoker`. Этот объект порождается из операции `Operation` и тесно с ней связан. Кроме того, этот объект содержит в себе код шаблона, который можно вызывать принудительно с помощью специальных методов. Как уже наверное понятно, этих методов пять и они вызывают на выполнение соответствующий обработчик события.

- **FireOnApply** – вызывает обработчик `OnApply`.
- **FireOnEdit** – вызывает обработчик `OnEdit`.

- **FireOnRecalc** – вызывает обработчик OnRecalc.
- **FireBeforeSave** – вызывает обработчик BeforeSave.
- **FireAfterSave** – вызывает обработчик AfterSave.

Вспомогательные объекты

Все вспомогательные объекты являются общими для всех приложений семейства *Акцент* и хранятся в библиотеке *Impact Common 7.4*.

Объект Map

В программах семейства *Акцент* каждый модуль, форма, диалог или электронная таблица является независимым объектом. Этот подход несколько отличается от других систем программирования, в которых существует понятие приложения. Каждый элемент системы не зависит от других и выполняется в своем собственном контексте. Однако, такой подход не предоставляет простых способов передачи информации между элементами приложения.

Для обеспечения этой возможности используется следующий подход:

В каждой форме, диалоге или таблице (но не в модуле) имеется специальный объект Map. Map можно рассматривать как массив пар *«имя – значение»* (такие объекты часто называют ассоциативными массивами). Имя –это просто текстовая строка. Доступ к значениям осуществляется по именам. После того, как объект создан, объект Map доступен вызывающей программе через свойство Map. Вызывающая программа может записать в Map любые значения. Эти значения будут доступны для запущенной программы.

Приведем простой пример. Допустим, что из формы нужно показать диалог. Диалог должен использовать текущую операцию формы для каких-то действий. Диалог запускается при нажатии кнопки с именем cmdDlg.

Внутри формы, обработчик события OnClick для кнопки cmdDlg будет таким:

```
' Обработчик события OnClick
Sub cmdDlg_OnClick()
   Dim dlg
   ' Создадим объект Dialog
   Set dlg = App.CreateDialog("dialog1.dlg")
   ' Запишем в Map диалога ID с именем   "DocID"
   dlg.Map("DocID") = op.ID
```

```
    Покажем диалог на экране
    ' при этом начнет работать программа диалога
    dlg.DoModal
End Sub
```

Внутри диалога все действия с операцией будут в обработчике события `OnLoad`:

```
' Обработчик события OnLoad
Sub Dialog_OnClick()
    Dim op
    ' Создадим операцию по ID из Map
    Set op = Workarea.Operation(Map("DocID"))
    ' выполняем какие-то действия с операцией
End Sub
```

Основные свойства объекта `Map`:

- **`Count`** – количество элементов.
- **`IsEmpty`** – возвращает `True`, если карта пуста.
- **`Item`** – возвращает или устанавливает значение по ключу. Аргументом свойства является текстовая строка, представляющая собой ключ.

Основные методы объекта `Map`:

- **`Exists`** – проверяет, существует ли элемент с заданным ключом.
- **`Remove`** – удаляет ключ (и значение тоже) из карты.
- **`RemoveAll`** – полностью очищает объект `Map`.

Содержимое объекта `Map` может быть представлено в виде XML. Для этого используется свойство `XML`. При чтении свойства, содержимое объекта сериализуется в XML и возвращается в виде текстовой строки. При записи – делается попытка разбора (parsing) XML-строки и из нее восстанавливается содержимое объекта. Обратите внимание, что свойство XML не поддерживает десериализацию карт, содержащие объекты.

Объект Redirect

Объект `Redirect` используется для доступа к информации о расположении файлов, которая хранится в файлах с расширением `ARD`. Объект соз-

дается с помощью функции CreateLibObject с предопределенным именем **"Redirect"**.

Основные свойства объекта Redirect:

- **AppPath,DbPath** – возвращают значения макросов $(AppPath) и $(DbPath), заданных в файле расположения.

Единственный метод объекта Redirect:

- **GetFullPath** – возвращает полный путь к файлу с учетом информации о расположении, заданной в файле *.ARD.

Объект Period

Объект Period используется для представления информации либо о текущем периоде, либо о периоде, в котором строятся отчеты. Объект возвращается одноименным свойством Period, которое есть у объекта Workarea и у всех объектов, связанных с отчетами.

Основные свойства объекта Period:

- **Start,End** – возвращают или устанавливают дату начала и окончания периода.
- **Year** – возвращает или устанавливает текущий рабочий год.
- **Name**, **Title** – возвращают наименование и заголовок периода.
- **Valid** – возвращает True, если в период входит хотя бы один день.

Методы объекта Period:

- **SetDay**, **SetMonth**, **SetQuart** – устанавливают период соответственно в заданный день, месяц или квартал.
- **SetMonthRange** – устанавливает период в заданный диапазон месяцев текущего года.
- **Browse** – показывает на экране стандартный диалог выбора периода.

Объект PageSetup

Объект PageSetup используется для управления свойствами печатной страницы. Объект содержится в формах (объект FrmPage) и электронных таблицах (объект ShtSheet). Следует отметить, что каждая страница

формы и лист таблицы имеет собственный объект PageSetup. Ссылка на объект возвращается одноименным свойством PageSetup.

Основные свойства объекта PageSetup:

- **LeftMargin,TopMargin,RightMargin,BottomMargin** – поля печатной страницы. Задаются в десятых долях миллиметра.
- **Orientation** – Задает ориентацию печатной страницы. Задается следующими константами:
  acDefault по умолчанию (как в принтере)
  acPortrait портретная
  acLandscape альбомная
- **FitToPage** – устанавливает признак того, что область печати нужно масштабировать так, чтобы она полностью поместилась на одной печатной странице (с учетом полей). Свойство имеет смысл только для электронных таблиц. Для форм значение свойства игнорируется.
- **LeftHeader,CenterHeader,RightHeader,LeftFooter,CenterFooter,RightFooter** – задают соответствующе части верхнего (Header) и нижнего (Footer) колонтитулов. В тексте колонтитулов допустимо использование макросимволов, которые при выводе на печать заменяются соответствующими значениями.
  &(Page) номер страницы
  &(Pages) количество страниц
  &(Date) дата печати
  &(Time) время печати
  &(Tab) заголовок страницы
  &(File) имя файла
  &(Company) наименование фирмы
  &(User) имя пользователя

Объект PageSetup поддерживает возможность более тонкого управления печатной страницей. Подробная информация об этом объекте приведена в электронной документации.

Объекты Toolbar и ToolbarButton

Эта группа объектов предназначена для построения пользовательских панелей инструментов в формах и электронных таблицах. Сама панель инструментов представлена объектом Toolbar, а каждая кнопка на ней

объектом ToolbarButton. Объекты нельзя создать, вместо этого у формы (объект Form) и таблицы (объект ShtBook) имеется одноименное свойство Toolbar, возвращающее ссылку на панель инструментов.

Сам объект Toolbar напоминает коллекцию. В нем также имеется свойство Item, возвращающее ссылку на кнопку панели (ToolbarButton).

Каждая кнопка панели инструментов идентифицируется целым числом, называемым *командой*. Когда пользователь щелкает по кнопке, родительский объект (форма или электронная таблица) получает специальное событие OnBarClick, в которое передается команда нажатой кнопки.

Если известна команда, то можно получить ссылку на саму кнопку, воспользовавшись свойством ItemByCommand объекта Toolbar.

Основные свойства объекта ToolbarButton:

- **Caption** – текст, выводимый на кнопке.
- **Command** – команда кнопки. Это просто целое число от 1 до 999, которое будет передаваться в обработчики событий OnBarClick и OnBarDropDown.
- **DropDown** – признак, говорящий о том, что кнопка «выпадающая».
- **Enabled** – если свойство равно False, кнопка недоступна и отображается серым цветом.
- **Group** – если это свойство равно True, то левее кнопки изображается вертикальная черта, означающее начало группы.
- **Image** – номер картинки, изображаемой на кнопке. Номер это индекс в системном списке картинок.
- **Popup** – текст меню, которое появится при нажатии на стрелку. Работает только, если свойство DropDown равно True.
- **ToolTip** – текст подсказки, которая появляется при наведении курсора мыши на кнопку.
- **Visible** – если это свойство равно False, то кнопка невидима и недоступна.
- **Tag** – произвольная текстовая строка, связанная с кнопкой.

Основные свойства самой панели инструментов Toolbar:

- **Count** – количество кнопок на панели инструментов.
- **Item** – возвращает кнопку по индексу.
- **ItemByCommand** – возвращает ссылку на кнопку (объект `ToolbarButton`) по заданной команде.

Панель инструментов `Toolbar` имеет следующие методы:

- **Create** – создает на панели новую кнопку.
- **Refresh** – просто обновляет изображение кнопки.

Объект WaitCursor

Это очень простой объект. Он используется для того, чтобы показать курсор в виде «песочных часов». Для того чтобы не следить за восстановлением курсора в исходное положение, механизм управления курсором сделан в виде объекта, а не в виде встроенной функции. Это сделано потому, что объекты автоматически уничтожаются при выходе из области видимости.

Объект не имеет никаких методов и свойств, а курсор мыши будет иметь вид «песочных часов» все время, пока существует объект. Таким образом, наиболее удобно создавать этот объект внутри процедуры, выполняющей длительную операцию. При завершении процедуры объект выходит из области видимости и автоматически уничтожается, что приводит к возврату вида курсора к исходному состоянию.

Если нужно вернуть курсор в исходное состояние вручную, объект можно принудительно уничтожить, присвоив переменной значение `Nothing`.

Объект создается одноименной встроенной функцией `WaitCursor()`.

Пример

```
Sub Do_Long_Operation
   Dim wc
   Set wc = WaitCursor
   ' очень долго делаем что-то
   ' при выходе из Sub, объект wc удалится
End Sub
```

Объект Meter

Этот объект представляет собой обычный индикатор прогресса и используется для того, чтобы информировать пользователя в процессе выполнения длительных действий.

Индикатор открывается в строке состояния активного окна. Поэтому, не рекомендуется использовать индикатор в диалогах. Если это сделать, индикатор появится, но может быть не виден пользователю. Для использования в диалогах имеется специальный элемент управления `DlgMeter`, который делает то же самое, что и `Meter`.

Создается объект одноименной глобальной функцией `Meter()`.

Объект `Meter` очень прост и имеет всего несколько свойств и методов.

- **`Open`** – открывает индикатор прогресса. Имеет необязательные аргументы `Min` и `Max`, которые задают минимальное и максимальное положение индикатора.
- **`Close`** – закрывает индикатор. После закрытия его можно открыть снова.
- **`Pos`** – текущая позиция индикатора.
- **`Min,Max`** – минимальное и максимальное положение позиции индикатора.
- **`Caption`** – текстовая строка, которая выводится перед индикатором и обычно описывает выполняемое действие.
- **`StepIt`** – наращивает значение индикатора на один шаг.
- **`SetStep`** – устанавливает шаг изменения позиции индикатора.
- **`OffsetPos`** – наращивает позицию индикатора на заданное значение.

Объект PopupMenu

Объект `PopupMenu` используется для создания всплывающих меню. Объект создается с помощью функции `CreateLibObject` с предопределенным именем **`"PopupMenu"`**.

Каждая команда меню характеризуется текстом и командой. Команда представляет собой целое число от 1 до 999. Когда пользователь выбирает

команду из меню, то для родительского объекта, вызвавшего меню (например, для формы или диалога) генерируется событие OnPopup.

Основные свойства и методы для объекта PopupMenu:

- **CreateMenu** – создает новое меню. Имеет один необязательный аргумент Text, позволяющий задать строку, из которой конструируется меню.
- **AppendMenu** – добавляет к меню новый пункт.
- **Track** – показывает меню на экране. После выбора команды меню закрывается и генерируется событие OnPopup.
- **Enabled** – запрещает или разрешает конкретную команду меню
- **Checked** – отмечает или снимает отметку с конкретной команды меню.

Совет. Показать всплывающее меню в форме или в диалоге можно и не создавая объект PopupMenu. Для этого можно воспользоваться методом ShowPopup. Однако в этом случае управлять разрешением/запрещением команд меню или устанавливать отметки будет невозможно.

Объект FileDialog

С помощью объекта FileDialog можно показать пользователю стандартные диалоги *Windows* «Открыть файл» и «Сохранить файл».

Объект создается с помощью функции CreateLibObject с предопределенным именем **"FileDialog"**.

Основные свойства объекта FileDialog, которые нужно установить до вызова диалога:

- **Caption** – заголовок окна диалога.
- **DefaultName, DefaultExt** – имя и расширение файла по умолчанию.
- **FileMustExists** – если значение этого свойства равно True, то выбрать несуществующий файл будет невозможно.
- **InitialDir** – путь к папке, которая должна быть текущей во время открытия диалога.

- **Filter** – строка фильтра для выбора имен файлов. Подробности см. в справочной системе.

Свойства объекта FileDialog, которые выдаются после получения результата:

- **PathName** – полный путь к выбранному файлу.
- **FileName** – имя выбранного файла.
- **FileTitle** – имя выбранного файла без расширения.

Методы объекта FileDialog служат для показа диалога на экране:

- **ShowOpen** – показывает диалог открытия файла.
- **ShowSave** – показывает диалог сохранения файла.

Объект System

Объект System служит для доступа к информации о текущем компьютере и локальной сети, на котором выполняется приложение, а также для работы с оболочкой операционной системы.

Объект создается с помощью функции CreateLibObject с предопределенным именем **"System"**.

Основные свойства объекта System:

- **ComputerName** – имя локального компьютера, под которым он известен в локальной сети.
- **UserName** – имя текущего пользователя, под которым он вошел в сеть. Обратите внимание, что это не имя пользователя в *Акцент*, а имя, под которым текущий пользователь известен операционной системе.
- **TempFolder** – путь к папке Windows, которая используется для хранения временных файлов.

Методы объекта System для управления приложениями:

- **Run** – запускает заданное приложение, передавая ему командную строку.
- **AppActivate** – активизирует заданное приложение, помещает его окно на передний план и устанавливает на него фокус ввода.

Методы объекта System для работы с файловой системой:

- **CopyFile** – копирует заданный файл в новое место, возможно с перезаписью существующего файла.
- **DeleteFile** – позволяет удалить заданный файл.

Объект WinAPI

Объект WinAPI предоставляет доступ к некоторым часто встречающимся функциям интерфейса прикладного программирования *Win32 API*.

Объект нельзя создать с помощью CreateObject. Вместо этого воспользуйтесь функцией CreateLibObject с предопределенным именем **"WinAPI"**.

Ниже перечислены основные методы объекта WinAPI. За более подробными сведениями обращайтесь к электронной справочной системе.

- **AnsiToOem,OemToAnsi** – преобразовывает текстовую строку ANSI(Windows) или в OEM(DOS) кодировки. Очень полезны при работе с устаревшими программами, а также с оборудованием, не поддерживающим Windows (матричные принтеры, кассовые аппараты и т.д.)
- **WinHelp** – вызывает справочную систему Windows. В качестве аргументов, метод принимает имя справочного файла (с расширением HLP), команду, которую нужно выполнить и, возможно, контекстный идентификатор, определяющий нужную тему справки.
- **HtmlHelp** – вызывает справочную систему в новом формате HtmlHelp. В метод передается имя справочного файла (с расширением CHM), выполняемая команда и, возможно, дополнительная информация для программы справки.
- **NewGUID** – создает новый глобальный уникальный идентификатор (GUID). Такие идентификаторы являются гарантированно уникальными и используются при репликации баз данных. Можно использовать произвольно для получения гарантированно уникальных ключей.
- **RegWrite** – позволяет записать в реестр (*Registry*) произвольную строку. Используется для хранения разнообразных пользовательских установок. Обратите внимание, что метод записывает строку в соб-

ственный ключ реестра. Его нельзя использовать для записи в реестр произвольных ключей.

- **RegRead** – читает из реестра текстовую строку, записанную туда методом RegWrite.
- **GetKeyState** – возвращает состояние заданной аргументом клавиши. Если клавиша нажата, возвращает True. Обычно используется для опроса клавиш-модификаторов, таких как Ctrl, Alt и Shift.
- **IsEscPressed** – метод асинхронно (в процессе работы программы) проверяет, была ли нажата клавиша Esc. Фактически, метод возвращает True, если в буфере клавиатуры содержится символ **Escape**. Метод позволяет организовать прерывание длительных процессов. Для этого, внутри цикла нужно периодически вызывать IsEscPressed, и, если он вернул True, то завершать операцию.

Объект PopupToolTip

Объект PopupToolTip представляет собой окно всплывающей подсказки с возможностью программного управления.

Объект нельзя создать с помощью CreateObject. Вместо этого воспользуйтесь функцией CreateLibObject с предопределенным именем **"PopupToolTip"**. Основное назначение объекта – вывод всплывающих подсказок в процессе выделения каких-либо элементов в системе. Например, подсказку можно открывать (и перемещать) в обработчике события OnRangeTrack, а закрывать в обработчике события OnRangeSelect.

Ниже перечислены основные свойства и методы объекта PopupToolTip. За более подробными сведениями обращайтесь к электронной справочной системе.

- **Caption** – текст всплывающей подсказки.
- **Open** – открывает окно подсказки.
- **Close** – закрывает окно подсказки.
- **Move** – перемещает окно в другое положение.

Объект DHTMLDialog

Объект DHTMLDialog позволяет показать *HTML*-страницу в виде диалога. Диалог может быть как модальным (для продолжения работы с прило-

жением нужно завершить работу с диалогом), так и немодальным (можно продолжать работу не закрывая диалог). Код скриптов, которые содержатся на *HTML*-странице позволяет получить доступ к объектной модели *Акцент*.

Объект создается с помощью функции `CreateLibObject` с предопределенным именем **`"DHTMLDialog"`**.

Перед показом диалога на экране, в него можно передать аргументы (методом `SetArgument`). Существует более короткий способ вызвать такой диалог из программы (этот способ функционально ограничен). Для этого можно использовать метод `ShowDHTMLDialog` объекта `Application`.

Основные свойства объекта `DHTMLDialog`:

- **`Caption`** – заголовок окна диалога.
- **`HTMLDocument`** – возвращает ссылку на объект `document` объектной модели *DHTML DOM*. Позволяет управлять HTML-страницей из программы на *VBScript*.
- **`Width, Height`** – ширина и высота окна диалога.
- **`IsDialogOpen`** – позволяет определить, открыто ли окно немодального диалога.
- **`Resizable`** – определяет, можно ли будет изменять размеры диалога.
- **`URL`** – основное свойство, которое задает *URL* отображаемой страницы.

Для управления диалогом используются методы:

- **`DoModal`** – показывает диалог в модальном режиме.
- **`CreateModeless`** – открывает диалог в немодальном режиме.
- **`CloseModeless`** – закрывает немодальный диалог.
- **`SetArgument`** – позволяет передать в программу диалога значения аргументов.

Более подробная информация о работе с *HTML* в среде Акцент 7.4 приведена в главе **XII** (страница 266)

Объект DHTMLAssistant

Объект DHTMLAssistant представляет собой окно (панель), которое можно вывести в правой части основного окна приложения, формы или электронной таблицы. Обычно это окно используется для вывода динамических подсказок, сообщений об ошибках, советов и т.д. Формат отображения информации в окне ассистента – динамический *HTML*. Программный код на HTML-странице имеет доступ к объектной модели вызвавшего его элемента *Акцент*.

Ссылка на объект может быть получена с помощью свойства Assistant, которое имеется у следующих объектов: Workarea, Form, ShtBook.

Основные свойства объекта DHTMLAssistant:

- **CanGoBack,CanGoForward** – определяет, возможна ли навигация назад или вперед.
- **HTMLDocument** – возвращает ссылку на объект document объектной модели *DHTML DOM*. Позволяет управлять HTML-страницей из программы на *VBScript*.
- **Visible** – определяет, видимо ли окно ассистента.

Объект DHTMLAssistant управляется следующими методами:

- **Navigate** – осуществляет переход к заданному *URL*.
- **GoBack,GoForward** – выполняет переход назад или вперед по стеку браузера.
- **SetArgument** – позволяет передать в программу HTML-страницы значения аргументов.

Глава VI.

Отчеты

В этой главе описана работа с объектами отчетов *Акцент*. Приводится описание свойств и методов отчетов, а также примеры их использования.

Все отчеты в системе, которые можно построить с использованием интерфейса пользователя, доступны также и в виде объектов. Все объекты отчетов имеют префикс `Rep` и хранятся в системной библиотеке «`Impact Reports 7.4`».

Все отчеты, доступные в системе, перечислены в следующей таблице.

Элемент	Назначение
`RepCommon`	Объединение `RepBalance` и `RepChess`.
`RepChess`	Шахматная ведомость.
`RepBalance`	Оборотно-сальдовая ведомость.
`RepCross`	Шахматная ведомость по аналитике.
`RepSubList`	Выписка из журнала операций.
`RepDocList`	Реестр документов.
`RepBankList`	Выписки из банка.
`RepCashBook`	Кассовая книга.
`RepSynth`	Синтетическая ведомость по счету.
`RepTurn`	Оборотно-сальдовая ведомость по счету.
`RepGLedger`	Главная книга
`RepJournal`	Шахматный журнал-ордер или ведомость
`RepWizard`	Отчет, построенный мастером
`RepTax`	Налоговый отчет, собирающий информацию по меткам проводок

Объекты отчетов можно использовать как для получения отчетной информации, так и для вывода их в электронную таблицу с последующей обработкой.

Для создания отчетов используется метод `CreateReport` объекта **`Workarea`**. Этот метод получает единственный аргумент, задающий имя класса объекта и возвращает ссылку на объект отчета.

Некоторые отчеты представляют собой коллекции элементов отчетов. В этом случае элемент коллекции имеет такое же имя, как и отчет, но с добавлением суффикса `Item`. Например, реестр документов (`RepDocList`) можно рассматривать как коллекцию строк, каждая из которых представлена объектом `RepDocListItem`.

Общие свойства и методы отчетов

Все объекты отчетов обладают общими свойствами. Во-первых, все они имеют внутри объект, хранящий в себе период для построения отчета. В момент создания отчета, значение периода устанавливается равным текущему рабочему периоду. Ссылка на период возвращается свойством `Period`, которое есть у всех отчетов.

Во-вторых, перед тем как использовать данные отчета, его нужно построить. Это делается методом `Build`, который имеется у всех отчетов. Метод возвращает `True`, если отчет успешно построен и `False` в противном случае.

После того как отчет построен, данные из него можно использовать. Можно вывести отчет в электронную таблицу, получив на нее ссылку. Такой способ позволяет выполнить какую либо дополнительную обработку отчета перед тем, как показать его пользователю.

Вывод отчета в таблицу выполняется с помощью метода `MakeSheet`, который имеется у всех отчетов.

Другие общие свойства отчетов следующие:

- **`PlanID`** – для отчетов, которые строятся по плану счетов можно задать идентификатор (`ID`) плана счетов. Если он не указан, используется план счетов по умолчанию, заданный в свойствах базы данных.
- **`AccID`** – для отчетов, которые строятся по одному счету, перед его построением обязательно нужно указать идентификатор счета.
- **`CurID`** – если нужно построить отчет в валюте, то перед его построением нужно задать `ID` валюты. Если она не указана, то считается равной 1 и отчет будет построен в базовой валюте.

- **Caption** – заголовок отчета. Если его не задать, будет использоваться стандартное значение (для каждого отчета оно свое).

Объекты отчетов

Отчет RepCommon

В некотором смысле отчет является исключением из общих правил, приведенных выше. Он не имеет методов `Build` и `MakeSheet` и вывести его в таблицу невозможно.

Основное предназначение этого отчета – заполнение стандартных бланков типа «Баланса предприятия» или «Отчета о финансовых результатах».

В таких бланочных отчетах чаще всего нужна информация из оборотно-сальдовой и шахматной ведомостей, причем иногда она должна быть сгруппирована по месяцам и по субсчетам. Именно эту информацию и выдает отчет `RepCommon`.

Для того чтобы правильно заполнить бланк, нужно просто создать отчет в модуле электронной таблицы, а в формулах ячеек записать ссылки на соответствующие свойства отчета.

Основные свойства отчета `RepCommon` следующие:

- **StartDb, StartCr** – возвращает значения остатков на начало периода по счету за заданный месяц.
- **TurnDb, TurnCr** – возвращает значения оборотов за период по счету за заданный месяц.
- **EndDb, EndCr** – возвращает значения остатков на конец периода по счету за заданный месяц.

Все эти свойства принимают три аргумента. Сначала задается счет, причем можно указывать как код счета в виде текстовой строки, так и его идентификатор в базе данных (в виде числа). Затем можно указать месяц, за который нужно получить данные. Если месяц не указан, используется текущий период. После этого можно указать дополнительный флаг, указывающий на то, что нужно учесть информацию по субсчетам.

Более подробная информация об отчете `RepCommon` приведена в электронной справочной системе.

Отчет RepChess

Отчет RepChess представляет собой классическую шахматную ведомость. Основное ее отличие от обычной заключается в том, что ее можно транспонировать. Это означает, что в верхней части отчета можно размещать как дебет, так и кредит счетов. Режим вывода задается свойством Transpose. Если оно равно False (по умолчанию), то отчет отображается как обычно – дебет слева, кредит сверху. Если значение свойства равно True, то дебет будет сверху и кредит слева.

Отчет строится по конкретному плану счетов (задается свойством PlanID). Возможно построение отчета в валюте. В этом случае нужно установить значение свойства CurID.

Имеется возможность ограничить счета, которые должны попасть в отчет. В этом случае получится выборка из полной шахматной ведомости. Ввести ограничения позволяет метод Restrict. Этот метод имеет два необязательных аргумента, которые задают списки счетов по дебету и по кредиту. Списки представляют собой текстовые строки, в которых через запятую перечислены коды счетов. Только эти счета попадут в отчет.

После того как отчет построен (напомним, что это делается с помощью метода Build), свойства CountDb и CountCr возвращают количество счетов по дебету или по кредиту, которые попали в отчет.

Другие свойства отчета RepChess.

- **Sum** – возвращает сумму из одной ячейки отчета. Имеет два аргумента, задающие индексы по дебету или по кредиту счета. Индексы могут изменяться от 1 до CoundDb или CountCr соответственно.
- **AccDb, AccCr** – возвращают счета из дебетовой или кредитовой частей отчета по индексу. Свойства возвращают ссылку на объект Account, представляющий собой бухгалтерский счет.
- **TotalDb, TotalCr** – возвращает общую сумму дебетовых или кредитовых оборотов для всего отчета.
- **Total** – возвращает общую сумму оборотов для всего отчета.

Отчет RepChess имеет одно вспомогательное свойство DataArray. Свойство возвращает двумерный массив результатов. Значение, которое возвращается этим свойством, фактически представляет собой переменную типа Variant, содержащую массив. Это свойство очень удобно при работе с внешними приложениями посредством механизма OLE-

Automation. Например, оно позволяет записать все данные отчета в *Microsoft® Excel* за одно обращение. Свойство имеет один необязательный аргумент, указывающий на то, нужно ли включать в массив итоговые значения.

Отчет RepBalance

Отчет RepBalance представляет собой стандартную оборотно-сальдовую ведомость. Ведомость строится по конкретному плану счетов. Идентификатор этого плана счетов задается свойством PlanID. По умолчанию значение свойства PlanID устанавливается равным плану счетов, который указан в свойствах базы данных как основной.

Если нужно построить оборотно-сальдовую ведомость в валюте, то перед построением отчета нужно установить свойство CurID, задающее идентификатор валюты. Отчет строится (заполняется данными) стандартным методом Build.

После построения отчет чем-то напоминает коллекцию. Каждая строка отчета представлена самостоятельным объектом RepBalanceItem. Сам отчет имеет свойство Count, которое возвращает количество строк и свойство Item, позволяющее получить доступ к строке отчета по индексу.

Другие свойства отчета следующие. Обратите внимание, что они относятся ко всему отчету. Другими словами это значения из строки «Итого».

- **StartDb, StartCr** – возвращают общую сумму остатков на начало периода по дебету или кредиту.
- **TunrDb, TurnCr** – возвращают общую сумму оборотов за период по дебету или кредиту.
- **EndDb, EndCr** – возвращают общую сумму остатков на конец периода по дебету или кредиту.
- **Count** – возвращает общее количество строк в отчете. Доступ к элементам отчета осуществляется с помощью свойства Item.
- **Item** – возвращает ссылку на элемент отчета, который представлен объектом RepBalanceItem. Свойство имеет целый аргумент, задающий номер строки, которую нужно получить. Номер строки изменяется от 1 до Count.

Объект RepBalanceItem

Объект RepBalanceItem – это одна строка отчета RepBalance. Объект хранит информацию об одном счете оборотно-сальдовой ведомости. Перечислим свойства объекта RepBalanceItem.

- **Account** – объектное свойство, которое возвращает ссылку на бухгалтерский счет (объект Account).
- **StartDb**, **EndDb**, **TurnDb**, **TurnCr**, **EndDb**, **EndCr** – свойства типа Currency, которые возвращают суммы на начало, обороты и суммы на конец.
- **BadStartSaldo**, **BadEndSaldo** – свойства возвращают True, если вид начального или конечного сальдо счета не совпадает с типом счета, заявленного в плане счетов. Например, если для активного счета получится кредитовое сальдо, то свойства вернут True.

Отчет RepCross

Отчет RepCross представляет собой шахматный отчет, в котором обороты по заданным счетам за период разворачиваются в таблицу по двум аналитическим показателям.

В качестве аналитических показателей могут выступать документы, их номера, корреспонденты или объекты учета.

Для отчетов, включающих в качестве аналитических показателей объекты учета, имеется возможность отображать не только суммы, но количество объектов учета, используемых в проводках.

Свойства и методы, которые нужно задать перед построением отчета.

- **Period** – период для построения отчета. По умолчанию равен текущему рабочему периоду.
- **PlanID** – план счетов, по которому будет строиться отчет. Значение по умолчанию совпадает с основным планом счетов, указанным в свойствах базы данных.
- **CurID** – валюта, в которой строится отчет. По умолчанию -- базовая валюта.
- **RowMode**, **ColMode** – типы аналитических показателей по колонкам и строкам отчета. Возможны следующие значения.

Значение	Показатель
0	Наименование операции
1	Номер документа
2	Корреспондент (дебет)
3	Корреспондент (кредит)
4	Объект учета

- **SetAccDb**, **SetAccCr** – эти два метода задают счета, обороты по которым включаются в отчет. Счета задаются в виде текстовой строки, которая содержит список кодов счетов, разделенных запятыми.

После того как отчет будет построен методом **Build**, следующие свойства обеспечивают получение выбранных данных.

- **Rows**, **Columns** – возвращают количество строк и колонок, которые получились в результате построения отчета.
- **Sum**, **Qty** – возвращают сумму или количество в ячейке отчета. Свойства имеют два аргумента, задающих строку и колонку, по которым нужно получить значение. Номер строки меняется от 1 до `Rows`, а номер колонки от 1 до `Columns`.
- **RowSum**, **ColSum** – возвращают общую сумму по строке и колонке. Свойства имеют один обязательный аргумент, который задает индекс строки или колонки.
- **RowText**, **ColText** – возвращают показатель по строке или по колонке в виде текстовой строки. Для корреспондентов и объектов учета это будет наименование.
- **RowValue**, **ColValue** – возвращают внутреннее значение показателя по строке или по колонке. Для корреспондентов и объектов учета это будет идентификатор соответствующего объекта.
- **Total** – возвращает общую сумму оборотов, которые попали в отчет. Это будет сумма значений, которые вернут свойства `RowSum` или `ColSum`, если перебрать все строки или колонки отчета.

Отчет RepSubList

Отчет `RepSubList` представляет собой выписку из журнала операций (фактически это просто список проводок), построенную по заданному критерию за период.

Как и все остальные, отчет создается методом `CreateReport` с предопределенным именем `"RepSubList"`.

После построения отчета фактически он будет представлять собой коллекцию объектов `RepSubListItem`, каждый из которых представляет одну проводку, попавшую в отчет.

Перечислим свойства, которые нужно задать перед построением отчета.

- **`Period`** – период для построения отчета. По умолчанию равен текущему рабочему периоду.
- **`Kind`** – Критерий выборки. Это вид элемента, по которому строится отчет. Задается следующими константами:

Константа	Значение	Тип элемента
`acFolder`	**`1`**	Папка
`acAccount`	**`2`**	Счет
`acAgent`	**`3`**	Корреспондент
`acEntity`	**`4`**	Объект учета
`acMisc`	**`5`**	Произвольная аналитика
`acBinder`	**`6`**	Подшивка
`acTemplate`	**`7`**	Шаблон
	`33`	Все операции в текущем периоде
	`39`	Список проводок для значения из налогового отчета.

- **`KindID`** – идентификатор элемента, по которому будет строиться отчет. Для типа `acFolder` возможно нулевое значение. В этом случае выписка будет строиться для документов, не входящих ни в какую папку.
- **`Caption`** – свойство возвращает или устанавливает заголовок для отчета.
- **`StrParam1`** – строковый параметр. Задает адрес в налоговом отчете, для которого нужно получить расшифровку. Имеет смысл только для `Kind = 39`.

После того как отчет успешно построен (методом `Build`), доступны следующие свойства.

- **`Count`** – количество элементов (строк) в отчете.

- **Item** – свойство позволяет получить элемент (строку) отчета по индексу. Индекс может изменяться от 1 до Count. Каждый элемент отчета представлен самостоятельным объектом RepSubListItem.
- **Total** – это свойство возвращает общую сумму всех проводок, попавших в отчет (без учета не проведенных).

Объект RepSubListItem

Объект RepSubListItem – это одна строка (проводка), которая содержится в отчете RepSubList.

Объекты RepSubListItem возвращаются с помощью свойства Item родительского объекта RepSubList. Свойство имеет один обязательный аргумент, задающий индекс возвращаемой строки. Индекс может изменяться от 1 до Count.

Основные свойства объекта RepSubListItem следующие:

- **AccDbID**, **AccCrID** – идентификаторы счетов по дебету и по кредиту.
- **AgFromID**, **AgToID** – идентификаторы корреспондентов «Кто» и «Кому».
- **EntID** – идентификатор объекта учета для проводки.
- **DocID** – идентификатор документа или операции, в который входит данная проводка.
- **Date** – дата проводки. Обратите внимание, что это именно дата проводки, а не дата документа, содержащего эту проводку.
- **Sum** – сумма по проводке.
- **Name** – наименование операции или документа, в который входит данная проводка.
- **DocNo** – строковое значение, представляющее собой номер документа, содержащего проводку.
- **FormID** – идентификатор формы документа.

Отчет RepDocList

Отчет RepDocList представляет собой реестр документов, построенных по определенным критериям.

Отчет создается методом `CreateReport` объекта `Workarea`.

После построения отчета он будет содержать коллекцию элементов, каждый из которых описывает одну строку отчета (один документ) и представлен объектом `RepDocListItem`. Доступ к строкам отчета осуществляется с помощью свойства `Item` по индексу. Индекс может изменятся от 1 до значения свойства `Count`.

Перед построением отчета нужно задать следующие свойства:

- **`Period`** – период для построения отчета. По умолчанию равен текущему рабочему периоду.
- **`Caption`** – свойство возвращает или устанавливает заголовок для отчета.
- **`Kind`** – Критерий выборки. Это вид элемента, по которому строится отчет. Задается следующими константами:

Константа	Значение	Тип элемента
`acFolder`	**`1`**	Папка
`acAccount`	**`2`**	Счет
`acAgent`	**`3`**	Корреспондент
`acEntity`	**`4`**	Объект учета
`acMisc`	**`5`**	Произвольная аналитика
`acBinder`	**`6`**	Подшивка
`acTemplate`	**`7`**	Шаблон

- **`KindID`** – идентификатор элемента, по которому будет строиться отчет. Для типа `acFolder` возможно нулевое значение. В этом случае выписка будет строиться для документов, не входящих ни в какую папку.

После того как отчет успешно построен (методом `Build`), доступны следующие свойства.

- **`Count`** – количество элементов (строк) в отчете.
- **`Item`** – свойство позволяет получить элемент (строку) отчета по индексу. Индекс может изменяться от 1 до `Count`. Каждый элемент отчета представлен самостоятельным объектом `RepDocListItem`.
- **`Total`** – это свойство возвращает общую сумму всех документов, попавших в отчет (без учета не проведенных).

Объект RepDocListItem

Объект RepDocListItem это одна строка реестра документов. Фактически объект описывает одну строку (документ) отчета RepDocList. Ссылка на объект возвращается с помощью свойства Item родительского объекта RepDocList. Свойство Item имеет обязательный аргумент – индекс объекта в списке. Индекс изменяется от 1 до Count.

Перечислим свойства объекта RepDocListItem.

- **ID** – идентификатор документа или операции.
- **Date** – дата документа. Обратите внимание, что это именно дата документа или операции, а не дата проводок, входящих в этот документ.
- **Sum** – сумма по документу.
- **Name** – наименование операции или документа.
- **Memo –** примечание к операции или документу.
- **Tag** – признак, связанный с операцией или документом.
- **DocNo** – строковое значение, представляющее собой номер документа.
- **FormID** – идентификатор формы документа.

Отчет RepBankList

Отчет RepBankList представляет собой обычную классическую выписку из банка. Объект очень похож на выписку из журнала операций, но выводит информацию в другом виде. При построении отчета операции всегда сортируются по дате.

Свойства объекта, которые нужно установить до построения отчета.

- **Period** – период для построения отчета. По умолчанию равен текущему рабочему периоду.
- **Caption** – свойство возвращает или устанавливает заголовок для отчета.
- **AccID** – идентификатор счета, по которому нужно построить отчет.
- **CurID** – идентификатор валюты отчета. По умолчанию равен единице, что означает построение отчета в базовой валюте.

После того как отчет построен методом Build, можно использовать его данные или вывести его в электронную таблицу (методом MakeSheet).

Как и все отчеты, содержащие несколько элементов, отчет RepBankList имеет свойство Count (количество элементов в отчете) и свойство Item, возвращающее элемент отчета (в данном случае это будет объект RepBankListItem) по индексу.

Дополнительно к получению элементов отчет имеет следующие свойства для доступа к результатам.

- **SumRem** – возвращает сумму остатка для отчета. Обратите внимание, что свойство возвращает остаток в виде одного значения, а не в виде двух сумм по дебету и кредиту. Если остаток кредитовый, значение будет отрицательным.

Объект RepBankListItem

Объект RepBankListItem – это одна строка отчета «Выписки из банка» (RepBankList).

Как и для всех элементов отчетов или коллекций, ссылка на объект возвращается с помощью свойства Item родительского объекта RepBankList. Индекс элемента, передаваемый в качестве аргумента свойства, может изменяться от 1 до Count.

Основные свойства объекта RepBankListItem следующие.

- **AccID** – идентификатор корреспондирующего бухгалтерского счета.
- **AgID** – идентификатор корреспондента. Обратите внимание, что возвращается только один корреспондент.
- **SumDb**, **SumCr** – сумма по дебету или по кредиту для строки отчета.
- **SumRem** – сумма остатка в этой строке отчета. Если остаток кредитовый, сумма будет отрицательной. Поскольку операции в отчете отсортированы по дате, суммы остатков будут отображаться «лесенкой».
- **Date** – дата проводки, которая представляет собой строку отчета.
- **DocID** – идентификатор документа или операции, содержащего проводку.

- **DocNo** – строковое значение, представляющее собой номер документа.
- **FormID** – идентификатор формы документа.
- **Name** – наименование документа или операции.

Отчет RepCashBook

Отчет `RepCashBook` представляет собой стандартную бухгалтерскую кассовую книгу. Отчет похож на выписку из журнала операций (объект `RepSubList`).

Работа с кассовой книгой имеет одну особенность. Кассовая книга строится за один день. Если сегодняшняя дата входит в период построения отчета, то отчет строится на сегодняшнюю дату. При установке любого другого периода кассовая книга будет строиться только за первый день этого периода.

Отчет имеет дополнительное свойство `AgID`, позволяющее ограничить выборку операций одним корреспондентом. Использование этого свойства позволяет получить кассовые книги по отдельным кассирам или подразделениям. Разумеется, для этого они должны быть указаны в проводках.

Обратите внимание, что такая кассовая книга недоступна для построения из интерфейса пользователя. Ее можно построить только из *VBScript*.

Возможно построение кассовой книги в валюте. Для этого используется свойство `CurID`, представляющее собой идентификатор валюты. Такую кассовую книгу тоже можно построить только из *VBScript*.

Перед построением отчета (методом `Build`) нужно указать следующие свойства.

- **Period** – период для построения отчета. По умолчанию равен текущему рабочему периоду. Если сегодняшняя дата входит в период, то отчет строится за сегодня, если нет – то за первый день периода.
- **Caption** – свойство возвращает или устанавливает заголовок для отчета.
- **AccID** – идентификатор счета, по которому нужно построить отчет.
- **CurID** – идентификатор валюты отчета. По умолчанию равен единице, что означает построение отчета в базовой валюте.

- **AgID** – идентификатор корреспондента. Дополнительное ограничение для проводок, попадающих в отчет.

После того как отчет построен, можно использовать его данные или вывести его в таблицу (методом `MakeSheet`).

Свойства, обеспечивающие доступ к данным, полученным в отчете следующие.

- **SumRem** – сумма остатка на конец периода. Если остаток кредитовый, сумма будет отрицательной.
- **TotalDb**, **TotalCr** – суммы оборотов по дебету и по кредиту для всего отчета.
- **Count** – количество элементов в отчете.
- **Item** – свойство возвращает ссылку на элемент отчета по его индексу. Элемент отчета представлен объектом `RepCashBookItem`. Индекс может изменяться от 1 до `Count`.

Объект RepCashBookItem

Объект `RepCashBookItem` – это одна строка отчета «Кассовая книга» (`RepCashBook`). Ссылку на объект можно получить с помощью свойства `Item` кассовой книги. Свойство имеет один целый аргумент, представляющий собой индекс нужной строки. Индекс может изменяться от 1 до `Count`.

Основные свойства объекта `RepCashBookItem` следующие:

- **AccID** – идентификатор корреспондирующего бухгалтерского счета.
- **AgID** – идентификатор корреспондента. Обратите внимание, что возвращается только один корреспондент.
- **SumDb**, **SumCr** – сумма по дебету или по кредиту для строки кассовой книги.
- **DocID** – идентификатор документа или операции, содержащего проводку.
- **DocNo** – строковое значение, представляющее собой номер документа.
- **FormID** – идентификатор формы документа.

- **Name** – наименование документа или операции.

Отчет RepSynth

Отчет RepSynth представляет собой схему счета по синтетическому учету. Схема строится по заданному счету за период.

В отличие от выписок из журнала, реестра документов и других похожих отчетов, внутри объекта отчета не содержится других объектов. Вся информация, полученная в результате построения отчета, доступна через свойства самого отчета.

Свойства отчета, которые нужно установить до его построения.

- **Period** – период для построения отчета. По умолчанию равен текущему рабочему периоду.
- **Caption** – свойство возвращает или устанавливает заголовок для отчета.
- **AccID** – идентификатор счета, по которому нужно построить отчет.
- **CurID** – идентификатор валюты отчета. По умолчанию равен единице, что означает построение отчета в базовой валюте.

После того, как все нужные свойства установлены (обратите внимание, что обязательно указывать только счет – свойство AccID), отчет можно построить с помощью метода Build.

После построения отчет будет содержать несколько строк. Их количество возвращается свойством Rows.

Свойства, которые относятся к строкам отчета, принимают в качестве аргумента номер (индекс) строки. Этот номер может изменяться от 1 до Rows. Эти свойства следующие:

- **RowDate** – дата, указанная в строке.
- **SumDb**, **SumCr** – сумма по дебету и кредиту, попавшая в указанную строку отчета.
- **AccDb**, **AccCr** – ссылки на счета по дебету и кредиту в строке отчета. Это ссылки на объект Account, представляющий собой бухгалтерский счет.
- **AccDbID**, **AccCrID** – идентификаторы счетов по дебету и кредиту. Фактически они дублируют свойства AccDb, AccCr.

Помимо перечисленных выше, объект имеет несколько свойств, относящихся ко всему отчету в целом. Это следующие свойства.

- **StartDb**, **StartCr** – суммы остатка на начало периода по дебету и кредиту.
- **TurnDb**, **TurnCr** – суммы оборотов за период по дебету и кредиту.
- **EndDb**, **EndCr** – суммы остатка на конец периода по дебету и кредиту.

Отчет RepTurn

Отчет `RepTurn` представляет собой оборотно-сальдовую ведомость по счету. Отчет строится по заданному счету. Выбираются обороты по этому счету и группируются по корреспондирующим счетам.

Перед тем как построить отчет, нужно задать следующие свойства.

- **Period** – период для построения отчета. По умолчанию равен текущему рабочему периоду.
- **Caption** – свойство возвращает или устанавливает заголовок для отчета.
- **AccID** – идентификатор бухгалтерского счета, по которому нужно построить отчет.
- **CurID** – идентификатор валюты отчета. По умолчанию равен единице, что означает построение отчета в базовой валюте.

После того как все нужные свойства установлены (обратите внимание, что обязательно указывать только счет – свойство `AccID`), отчет можно построить с помощью метода `Build`.

После того как отчет построен, следующие свойства позволяют получить информацию о данных, попавших в отчет.

- **Count** – количество строк в отчете.
- **StartDb**, **StartCr** – суммы остатка на начало периода по дебету и кредиту.
- **TurnDb**, **TurnCr** – суммы оборотов за период по дебету и кредиту.
- **EndDb**, **EndCr** – суммы остатка на конец периода по дебету и кредиту.

Отчет можно вывести в электронную таблицу с помощью метода **MakeSheet**. Этот метод возвращает ссылку на объект **ShtBook** – книга электронной таблицы. Используя эту ссылку, можно изменить отчет после вывода в таблицу, но до показа пользователю.

Отчет RepGLedger

Отчет `RepGLedger` представляет собой выписку из бухгалтерской «Главной книги», построенной по определенному счету за год.

Обратите внимание, что отчет всегда строится за целый год, независимо от установленного периода. Однако, как и все другие отчеты, он имеет свойство `Period`. В данном случае имеет смысл только год, установленный в периоде.

Свойства отчета, которые нужно установить до его построения (методом `Build`).

- **Period** – период для построения отчета. По умолчанию равен текущему рабочему периоду. Используется только значение установленного в периоде года.
- **Caption** – свойство возвращает или устанавливает заголовок для отчета.
- **AccID** – идентификатор счета, по которому нужно построить отчет.
- **CurID** – идентификатор валюты отчета. По умолчанию равен единице, что означает построение отчета в базовой валюте.

Обратите внимание, что обязательно указывать только счет – свойство `AccID`, все остальные свойства имеют значения по умолчанию.

После того как отчет построен, следующие свойства позволяют получить данные, попавшие в отчет.

- **StartDb**, **StartCr** – суммы остатка на начало периода по дебету и кредиту для всего отчета.
- **TurnDb**, **TurnCr** – суммы оборотов за период по дебету и кредиту для всего отчета.
- **EndDb**, **EndCr** – суммы остатка на конец периода по дебету и кредиту для всего отчета.
- **AccArray** – массив идентификаторов корреспондирующих счетов. Эти счета будут перечислены в верхней части отчета.

Обратите внимание. Использование свойств, которые возвращают массивы, имеет одну особенность. Для доступа к массиву нужно использовать промежуточную переменную. Если, например, написать

```
rep.AccArray(0)
```

то *VBScript* выдаст ошибку из-за того, что будет считать что (0) это аргумент свойства, а не индекс массива. При использовании промежуточной переменной все будет работать правильно.

```
Dim ar
ar = rep.AccArray
x = ar(0)
```

Кроме того, можно явно использовать две пары скобок. Например:

```
Rep.AccArray()(0)
```

Первая пара говорит о том, что свойство не имеет аргументов, а вторая пара представляет собой индекс в возвращаемом массиве.

Отчет RepJournal

Отчет `RepJournal` представляет собой шахматную ведомость или шахматный журнал-ордер, которые развернуты по одному из аналитических показателей. Ведомость строится по дебету счета, а журнал-ордер – по кредиту счета. Для построения отчета нужно задать счета в верхней части отчета (по кредиту для журнала и по дебету для ведомости). Корреспондирующие счета будут выбраны автоматически. Кроме того, задается тип аналитики, по которой нужно строить отчет.

Отчет содержит внутри себя отдельные объекты `RepJournalItem`, представляющие собой элементы отчета. В отличие от выписок и реестров, один элемент отчета представляет собой не одну строку, а несколько строк, связанных с одним корреспондирующим счетом.

Обратите внимание. В отличие от других, отчет строится для нескольких счетов, поэтому для него указывается идентификатор не счета, а плана счетов.

Перед построением отчета нужно установить следующие свойства.

- **`Period`** – период для построения отчета. По умолчанию равен текущему рабочему периоду.

- **Mode** – Вид аналитики для построения отчета. Задается следующими константами:

Константа	Значение	Тип аналитики
	0	Наименование операции
	1	Номер документа
	2	Корреспондент (дебет)
	3	Корреспондент (кредит)
	4	Объект учета

- **Caption** – свойство возвращает или устанавливает заголовок для отчета.
- **PlanID** – идентификатор плана счетов, по которому нужно построить отчет.
- **CurID** – идентификатор валюты отчета. По умолчанию равен единице, что означает построение отчета в базовой валюте.
- **Transpose** – режим построения отчета. Если свойство равно `False` (по умолчанию), то строится журнал (дебет слева, кредит сверху). Если свойство равно `True`, то строится ведомость (кредит слева, дебет сверху).
- **SetAcc** – это метод, задающий коды счетов в верхней части отчета. Обратите внимание, что для журнала (`Transpose = False`) это будут счета по кредиту, а для ведомости (`Transpose = True`) – счета по дебету. Коды счетов задаются в виде текстовой строки и разделяются запятыми. В отчет попадут все проводки за период, в которых участвуют указанные счета.

Как и все остальные, отчет строится методом `Build`. После этого становятся доступны следующие свойства.

- **Count** – количество групп строк, которые получились в результате построения отчета.
- **Columns** – количество столбцов в отчете. Каждый столбец показывает данные по одному корреспондирующему счету.
- **Item** – объектное свойство, возвращающее ссылку на один элемент отчета. Элемент отчета представлен свойством `RepJournalItem`. Свойство имеет один аргумент – индекс элемента отчета. Индекс может изменяться от `1` до `Count`.

Следующие свойства возвращают информацию по колонкам информационной части отчета. Они имеют один аргумент, задающий индекс колонки в отчете. Индекс может изменяться от 1 до `Columns`.

- **`Account`** – свойство возвращает ссылку на счет, по которому построена колонка отчета. Сам объект, возвращаемый свойством, представляет собой бухгалтерский счет (объект `Account`).
- **`SumStart`**, **`SumTurn`**, **`SumEnd`** – свойства, возвращающие остатки на начало периода, обороты за период и остатки на конец периода для всего отчета. Используются для заполнения строки **«Итого»**.

Кроме вышеперечисленных, отчет имеет еще три свойства, возвращающие общие суммы для всего отчета. Эти свойства не имеют аргументов.

- **`TotalStart`**, **`TotalEnd`** – свойства возвращают общую сумму остатков на начало и на конец периода для всего отчета.
- **`Total`** – сумма всех оборотов за период, которые попали в отчет.

Объект RepJournalItem

Объект `RepJournalItem` представляет собой элемент ведомости или журнала-ордера `RepJournal`. Это не строка отчета, а группа строк, связанных с одним корреспондирующим счетом. Ссылка на объект возвращается с помощью свойства `Item` родительского отчета.

Каждый элемент шахматного журнала или ведомости содержит информацию об одном корреспондирующем счете. Ссылка на этот счет (объект `Account`) возвращается помощью свойства `Account`. Один элемент отчета может содержать несколько строк. Количество этих строк возвращается свойством `Count`. Общая сумма оборотов по всему элементу отчета возвращается свойством `Total`.

У объекта имеется несколько свойств, возвращающих информацию о строках отчета. У всех таких свойств имеется один обязательный аргумент, задающий номер (индекс) строки внутри элемента отчета. Индекс может меняться от 1 до `Count`.

Перечислим свойства отчета, которые возвращают информацию об одной строке отчета.

- **`Sum`** – сумма по строке и колонке. Свойство имеет два аргумента – номер строки в элементе и номер колонки во всем отчете.
- **`ItemText`** – аналитический показатель в виде строки.

- **ItemValue** – аналитический показатель в виде значения.

Два последних свойства возвращают информацию об аналитическом показателе. Их смысл зависит от значения свойства Mode. Все возможные значения перечислены в следующей таблице:

Mode	Тип аналитики	ItemText	ItemValue
0	Операция	Наименование	Наименование
1	Номер документа	Номер документа	Идентификатор формы
2	Корреспондент (дебет)	Наименование корреспондента	Идентификатор корреспондента
3	Корреспондент (кредит)	Наименование корреспондента	Идентификатор корреспондента
4	Объект учета	Наименование объекта учета	Идентификатор объекта учета

Отчет RepWizard

Для построения произвольных отчетов в программах *Акцент* используется «Мастер отчетов». Мастер позволяет задать критерии построения отчета, способы группировки данных и вид отображения остатков и оборотов. Объект RepWizard включает всю функциональность, доступную из мастера отчетов.

Отчет всегда строится по одному счету, возможно с учетом субсчетов. Имеется возможность дополнительно ограничить информацию, которая должна попасть в отчет. Например, можно построить отчет по счету и корреспонденту. В этом случае в него попадут только проводки, проходящие по заданному счету и заданному корреспонденту. Возможно построение отчетов в валюте.

В отличие от большинства других, отчет, построенный с помощью объекта RepWizard, имеет не линейную (в виде массива или коллекции), а иерархическую (древовидную) структуру. Это означает, что каждый элемент отчета содержит в себе коллекцию точно таких же элементов, которые, в свою очередь, тоже содержат вложенные элементы. Уровень вложенности отчетов в программах *Акцент* ограничен четырьмя.

Схема объектов для отчета, который содержит два уровня группировки приведена на рисунке ниже.

Ошибка! Ошибка связи.

Рис. 2. Схема отчета RepWizard

На рисунке показано, что отчет `RepWizard` содержит корневой (`Root`) элемент `RepWizardItem`. Внутри него находятся два элемента первого уровня группировки. Внутри второго элемента первого уровня находятся два элемента второго уровня группировки. Все элементы отчета, включая корневой, имеют один и тот же тип – `RepWizardItem`. Перебирая эту структуру, можно получить данные обо всех элементах отчета.

Совет. Подобные приведенной иерархические структуры данных удобнее всего обрабатывать рекурсивными функциями. Рекурсивной называется такая функция, которая прямо или косвенно вызывает сама себя.

Перед построением отчета нужно задать способ его построения. Это делается с помощью следующих свойств и методов.

- **`Kind`** – тип элемента, по которому строится отчет. По умолчанию отчет строится по счету.
- **`KindID`** – идентификатор элемента (объект учета, корреспондент и т.д.) , по которому строится отчет. Если свойство `Recursive` равно `True`, то в отчет попадут все вложенные элементы.
- **`AccID`** – идентификатор счета, по которому строится отчет. Если тип `Kind` равен `acAccount`, то `AccID` равен `KindID`. Для всех остальных типов обязательно указывать и `KindID` и `AccID`.
- **`SubAcc`** – учитывать ли субсчета основного счета при построении отчета.
- **`Period`** – объектное свойство, определяющее период построения отчета. По умолчанию период для отчета равен текущему рабочему периоду.
- **`Caption`** – заголовок отчета.
- **`CurID`** – идентификатор валюты, в которой строится отчет. По умолчанию свойство равно 1, что означает базовую валюту.
- **`AddColumn`** – метод добавляет в отчет колонку аналитического показателя. Допустимо использовать до четырех показателей. Аргументы метода задают тип показателя, возможно дополнительные идентификаторы, заголовок колонки для показателя и дополнитель-

ные атрибуты, которые нужно отображать для данного показателя. Подробная информация об аргументах этого метода содержится в электронной справочной системе.

- **SetRemMode** – метод устанавливает способ вывода остатков на начало и конец периода. Допустимы следующие способы вывода:

Значение	Режим вывода
0	Не выводить вообще
1	Выводить только сумму
2	Развернутые остатки (по уровням)
3	Выводить только сальдо

- **SetTurnMode** – метод устанавливает способ вывода оборотов по дебету и кредиту. Допустимы следующие значения.

Значение	Режим вывода
0	Не выводить вообще
1	Выводить только общую сумму
2	Разворачивать по горизонтали (журнал)
3	Разворачивать по вертикали (ведомость)

Оба метода имеют дополнительный аргумент, позволяющий включить вывод количества и цен для объектов учета.

- **SetSubTotal** – метод позволяет включить и выключить отображение промежуточных итогов по заданному уровню. Метод имеет два аргумента, первый из которых задает номер уровня, а второй значение свойства. Максимальный номер уровня может быть на единицу меньше количества уровней, добавленных с помощью метода `AddColumn`.

- **Restrict** – метод позволяет ограничить корреспонденцию счетов, которые должны попасть в отчет. Счета по дебету и кредиту задаются в виде списка кодов через запятую.

- **ReduceLevel** – свойство указывает, нужно ли сворачивать сальдо для конкретного уровня.

- **ReduceTotal** – свойство указывает, нужно ли сворачивать общее сальдо для всего отчета.

- **Recursive** – учитывать ли вложенные элементы. Имеет смысл, только если отчет строится по аналитическому показателю. Например, если отчет строится по папке корреспондентов и `Recursive`

равно `True`, то в отчет попадут все корреспонденты, находящиеся в заданной папке.

После того как отчет построен (напомним, что это делается методом Build), информация о результатах, попавших в отчет может быть получена с помощью следующих методов и свойств.

- **`Root`** – объектное свойство только для чтения. Свойство возвращает ссылку на корневой узел дерева элементов отчета. Каждый узел дерева представляет собой объект `RepWizardItem`. Поскольку элементы отчета организованы иерархически, то свойства `Children` и `Child` позволяют перебрать все элементы отчета.
- **`AccArrayDb`**, **`AccArrayCr`** – свойства возвращают массивы идентификаторов корреспондирующих счетов, которые попали в дебетовую и кредитовую часть отчета. Не забывайте использовать промежуточную переменную для доступа к элементам массива (см. стр. 158).

Объект `RepWizard` имеет два дополнительных метода, которые позволяют записать сформированный алгоритм построения отчета в таблицу базы данных и прочитать его оттуда.

- **`Load`** – загружает описание отчета из базы данных. В качестве аргумента метода выступает идентификатор отчета. Идентификатор можно увидеть в заголовке окна свойств отчета.
- **`Save`** – метод сохраняет описание отчета в базе данных. В качестве аргументов выступают тип и идентификатор элемента, с которым нужно связать данный отчет. Метод возвращает идентификатор вновь созданного и записанного отчета.

Пара методов `Load` и `Save` очень полезна при автоматическом (программном) создании настроек.

Объект RepWizardItem

Объект `RepWizardItem` представляет собой один элемент отчета `RepWizard`. В отличие от других элементов отчетов, `RepWizardItem` это не строка отчета, а именно элемент. Поскольку данные в отчете `RepWizard` организованы в рекурсивную иерархическую структуру, то можно рассматривать его как узел дерева.

Объект имеет следующие свойства, предназначенные для навигации по вложенным элементам.

- **Children** – возвращает количество вложенных элементов внутри текущего элемента.
- **Child** – объектное свойство, которое возвращает ссылку на вложенный элемент по индексу. Индекс может изменяться от 1 до Children. Свойство аналогично Item, но его название дополнительно подчеркивает рекурсивную структуру данных.

Свойства объекта RepWizardItem, которые обеспечивают доступ к содержимому уровня отчета.

- **Value** – значение элемента отчета. Это свойство имеет тип Variant и возвращаемое значение зависит от типа колонки (напомним, что колонка добавляется в отчет методом AddColumn). Если, например, колонка представляет собой корреспондента, то возвращаемое значение будет его идентификатором. Если колонка – номер документа, то возвращаемое значение будет строкой номера документа и т.д.
- **StartDb**, **StartCr** – суммы остатка на начало периода по дебету и кредиту.
- **TurnDb**, **TurnCr** – суммы оборотов за период по дебету и кредиту.
- **EndDb**, **EndCr** – суммы остатка на конец периода по дебету и кредиту.
- **TurnDbAcc**, **TurnCrAcc** – суммы оборотов в корреспонденции с заданным счетом. Свойства имеют один аргумент, задающий идентификатор корреспондирующего счета. Список идентификаторов счетов в дебетовой и кредитовой частях отчета возвращается с помощью свойств AccArrayDb и AccArrayCr родительского отчета RepWizard.

Пример построения отчета RepWizard

В следующем примере строится оборотная ведомость по объектам учета по 281 счету. Помимо сумм, в обороты включается количество. После того, как отчет построен, он выводится в электронную таблицу.

```
' оборотная ведомость по 281 счету
' объявление переменных обязательно
Option Explicit
' переменная для хранения ссылки на отчет
Dim r
```

```
' создание самого объекта отчета
Set r = Workarea.CreateReport("RepWizard")
' добавим колонку. 5 – объекты учета
r.AddColumn 5
' обороты по дебету в строку + количество
r.SetTurnMode False, 2, 1
' обороты по кредиту в строку + количество
r.SetTurnMode True, 2, 1
' установка ID счета. ID получается по коду
r.AccID = Workarea.GetAccID("281")
' Построение отчета
If r.Build() Then
  ' Вывод отчета в электронную таблицу
  r.MakeSheet
Else
  MsgBox "Ошибка при построении отчета"
End If
```

Отчет RepTax

Отчет RepTax представляет собой специальный отчет, собирающий и группирующий информацию о проводках, помеченных специальным образом. Отчет не имеет внешнего представления (и, соответственно, не имеет метода **MakeSheet**). Обычно отчет RepTax используется для заполнения бланочных отчетов, таких, например, как «Декларация НДС» или «Декларация о прибыли». Обратите внимание, что информация, попадающая в отчет никак не связана с бухгалтерскими счетами, корреспондентами и прочей аналитической информацией. Фактически, отчет просто собирает и группирует проводки в соответствии с пометками проводок. Такие пометки обычно представляют собой просто «адрес» (номер строки и столбца в отчете) в таблице, куда должна попасть нужная сумма.

Поскольку бланочные отчетные формы строятся за фиксированные периоды, то, в отличие от остальных отчетов, при построении отчета **RepTax** задается не рабочий период, а рабочий год. После построения можно получить информацию за месяц, квартал, полугодие и весь год. Минимальная дискретность получаемой информации – месяц.

Перед тем как построить отчет, нужно задать следующие свойства.

- **Year** – год для построения отчета. По умолчанию равен году текущего рабочего периода.

- **TaxID** – свойство возвращает или устанавливает идентификатор налогового отчета, информацию по которому нужно собирать и группировать.

Обратите внимание, что отчет `RepTax` не имеет метода `Build`. Построение отчета выполняется автоматически при первом обращении к данным. Так сделано для того, чтобы упростить программирование формул в электронных таблицах.

Следующие свойства позволяют получить информацию о данных, попавших в отчет.

- **MonthValue** – сумма за заданный месяц по заданному адресу.
- **QuartValue** – сумма за заданный квартал по заданному адресу.
- **NumMonthValues** – сумма за заданное количество месяцев текущего года. Это свойство помогает упростить создание отчетов с «нарастающим итогом».
- **YearValue** – сумма за весь текущий год.

Глава VII.

Формы

В этой главе описана работа с формами *Акцент* и объектная модель форм. Приводится описание элементов управления, которые могут находиться на форме, их свойств, методов и событий.

Для взаимодействия с пользователем в программных продуктах семейства *Акцент* используются **формы**. Форма представляет собой электронный аналог бумажного документа и может использоваться как для ввода, так и для печати документов на принтере.

Использование для ввода информации форм, отображающих на экране привычные бумажные документы, позволяет сократить время обучения пользователей и значительно упростить работу с программой.

Форма может быть многостраничной. Кроме того, каждая страница формы может автоматически печататься на нескольких бумажных страницах, в зависимости от содержимого расположенной на форме таблицы. В этом случае поддерживается возможность печати, как заголовка формы, так и заголовков каждой из страниц.

С точки зрения программиста, форма представляет собой набор страниц, на каждой из которых расположены элементы управления.

Элементы управления на форме могут быть различных типов. Это текстовые метки, редакторы, прямоугольные области, линии, разнообразные селекторы и т.д. Очень мощными элементами управления являются таблицы.

Объектная модель для форм

На рисунке 3 приведена схема объектной модели форм *Акцент*. На самом верху иерархии находится объект `Form`, который представляет собой собственно саму форму.

В форме может быть несколько страниц. Каждая страница формы представлена объектом `FrmPage`.

Ошибка! Ошибка связи.

Рис. 3. Объектная модель для форм

На странице располагаются элементы управления. Хотя элементы управления и располагаются на странице формы, они все равно находятся в общем контексте всей формы. Это говорит, в частности, о том, что имена элементов управления на разных страницах не могут повторяться.

Каждый элемент управления представлен самостоятельным объектом, у которого, как и у любого объекта имеются свойства методы и события.

При взаимодействии с пользователем элементы управления могут генерировать различные события. Следует отметить, что все обработчики событий находятся в контексте самой формы и именуются стандартным способом в виде:

```
имяЭлемента_имяСобытия
```

Например, при нажатии командной кнопки с именем `cmdButton` система будет пытаться выполнить процедуру `cmdButton_OnClick`. Обратите внимание на то, что для того, чтобы можно было обрабатывать события от элемента управления, нужно чтобы элемент имел непустое имя.

Возможные типы элементов управления приведены в таблице ниже.

Элемент	Назначение
`FrmLabel`	Текстовая метка (не редактируемая)
`FrmLine`	Прямая линия
`FrmRect`	Прямоугольник
`FrmPicture`	Картинка
`FrmButton`	Командная кнопка
`FrmCheck`	Флажок
`FrmRadio`	Переключатель
`FrmEdit`	Текстовый редактор
`FrmSelector`	Текстовый редактор с кнопкой для выбора (селектор)
`FrmDate`	Элемент управления для ввода даты
`FrmComboBox`	Комбинированный список для выбора
`FrmGrid`	Таблица
`FrmScrollGrid`	Таблица с возможностью прокрутки
`FrmActiveX`	Элемент управления *ActiveX*, расположенный на форме.

Создание формы

В программах семейства *Акцент* создание объекта, содержащего ссылку на форму, возможно следующим способом. Форму можно создать вызовом метода `CreateForm` объекта `Application`.

Этот метод имеет следующий синтаксис.

Синтаксис

```
Set f = _
app.CreateForm(Frm,[OpID],[Fld],[Tml],[Flags])
```

Аргументы

`Frm` – Обязательный аргумент типа `Variant`. Если это целое число, то оно рассматривается как идентификатор формы, подключенной к проекту. Если это строка, то она рассматривается как имя файла диалога `*.AFM`. Если путь к файлу не указан, учитывается информация о расположении файлов из `*.ARD`.

`OpID` – Необязательно. Идентификатор операции, которая будет загружена в диалог.

`Fld,Tml` – Необязательно. Идентификаторы папки и шаблона для загруженной в форму операции.

`Flags` – Необязательно. Дополнительный набор флажков, который позволяет изменить поведение метода. Возможные значения флажков приведены в таблице ниже.

16 с/с	10 с/с	Назначение
`&H01`	1	Форма открывается в «невидимом» режиме. Показать форму можно, установив значение свойства `Visible` в `True`.

Не забывайте о том, что метод возвращает ссылку на объект, поэтому использование ключевого слова `Set` является обязательным.

Если в форму загружается бухгалтерская операция, то ссылку на нее можно получить с помощью свойства формы `CurrentOperation`.

Объект Form

Объект `Form` представляет собственно форму, включая основное окно и все страницы. Страницы формы (`FrmPage`) являются самостоятельными объектами и содержат внутри себя элементы управления. Кроме страниц,

в форме есть карта Map и пользовательская панель инструментов Toolbar. Ссылки на них возвращаются соответствующими свойствами.

Основные свойства объекта Form следующие.

- **ActivePage** – возвращает ссылку на текущую активную страницу.
- **Caption** – текстовая строка, определяющая заголовок окна формы.
- **Control** – обеспечивает доступ к элементу управления по имени.
- **CurrentOperation** – возвращает ссылку на текущую операцию (объект Operation), связанную с формой.
- **LockRecalc** – если это свойство имеет значение True, то пересчет формы не выполняется и события BeforeRecalc и AfterRecalc не возникают.
- **Map** – возвращает ссылку на карту Map.
- **ReadOnly** – если это свойство имеет значение True, то редактирование таблиц невозможно.
- **Page** – обеспечивает доступ к страницам формы. Работает как по индексам, так и по именам.
- **PageCount** – возвращает количество страниц формы.
- **ReadOnly** – если это свойство равно True, то редактирование формы запрещено.
- **ScreenWidth, ScreenHeight** – возвращают или устанавливают экранные размеры окна формы. При установке размеров, размеры панелей инструментов и строки состояния учитываются автоматически.
- **Toolbar** – обеспечивает доступ к пользовательской панели инструментов (объект Toolbar).
- **Visible** – если свойство равно True, то таблица видима, если False – то скрыта.
- **Zoom** – обеспечивает возможность показа формы в разном масштабе.
- **Assistant** – возвращает ссылку на объект DHTMLAssistant, который представляет собой окно Ассистента, связанного с формой.

Для выполнения действий со всей формой используются следующие методы.

- **Close** – закрывает окно формы.
- **Maximize, Minimize** – распахивает окно формы во весь экран или сворачивает его в значок.
- **FireUserEvent** – обеспечивает генерацию события `OnUser`.
- **Move** – перемещает окно таблицы по экрану.
- **Preview** – переводит форму в режим предварительного просмотра перед печатью.
- **Print** – выводит форму на печать. Имеется возможность показывать или не показывать пользователю диалог печати.
- **Recalc** – пересчитывает активный рабочий лист.
- **ResetFocus** – убирает фокус ввода со всех элементов управления и устанавливает его на саму форму.
- **ShowPopup** – позволяет показать на экране всплывающее меню. При выборе команды из такого меню генерируется событие `OnPopup`.
- **ExecBarTag** – рассматривает значение свойства `Tag` кнопки панели инструментов как инструкцию *VBScript* и выполняет ее. Предназначено для упрощения написания обработчика события `OnBarClick`.
- **RefreshScreen** – позволяет перерисовать (обновить) содержимое окна формы.
- **SetTimer** – устанавливает пользовательский таймер. Таймер периодически вызывает срабатывание события `OnTimer`.
- **HideVScroll, HideHScroll** – удаляют с экрана соответственно вертикальную и горизонтальную полосы прокрутки. При удалении горизонтальной полосы прокрутки также скрываются и ярлычки страниц формы.
- **ClientToScreen** – преобразовывает координаты из системы координат формы (0.1 мм) в координаты экрана (пиксели). Обычно используется совместно с объектами, которые используют адресацию в

экранных координатах (например всплывающая подсаказка PopupToolTip).

В процессе своей работы объект Form может получать и обрабатывать следующие события.

- **BeforePrint,AfterPrint** – возникают перед/после печати формы.
- **BeforeRecalc,AfterRecalc** – возникают перед/после пересчета формы. Позволяют контролировать процесс пересчета.
- **CanClose** – возникает при попытке закрыть форму. Имеется возможность запретить закрывать форму, либо, выполнив дополнительные действия, разрешить закрытие.
- **OnActivate** – возникает при активизации какой-либо страницы формы.
- **OnBarClick** – возникает при нажатии кнопки на пользовательской панели инструментов.
- **OnBarDropDown** – возникает при нажатии «выпадающей» кнопки на пользовательской панели инструментов.
- **OnPopup** – возникает при выборе команды из всплывающего меню, которое может быть вызвано методом ShowPopup или с помощью объекта PopupMenu.
- **OnLoad** – возникает после загрузки таблицы, но до появления ее на экране.
- **OnUser** – возникает при вызове метода FireUserEvent.
- **OnResize** – возникает, когда пользователь изменил размеры окна формы. Новые размеры передаются в обработчик событий.
- **OnHelp** – возникает, когда пользователь хочет получить справку по элементу управления или по кнопке панели инструментов. Это происходит при нажатии клавиши F1, выборе команды **«Что это значит?»** из всплывающего меню или при нажатии Shift+F1 и последующем указании курсором мыши на какой либо элемент управления. Чтобы вызвать справочную систему *Windows*, можно использовать метод WinHelp вспомогательного объекта WinAPI (стр. 136).

- **OnTimer** – возникает по истечении времени, которое задано в методе SetTimer.

Объект FrmPage

Объект FrmPage представляет собой одну страницу формы.

Внутри объекта FrmPage находятся все элементы управления. Кроме того, в этом объекте есть объект PageSetup, предназначенный для управления параметрами печатной страницы. Ссылка на него возвращаются соответствующим свойством PageSetup.

Основные свойства страницы FrmPage следующие:

- **Name** – имя страницы, под которым она известна в программе. Имя не может быть пустым. При добавлении страницы имя генерируется автоматически. Изменить имя во время выполнения программы невозможно.
- **Caption** – заголовок страницы. Заголовок выводится на ярлычке страницы в нижней части окна формы.
- **BackColor –** цвет фона страницы формы.
- **Visible** – признак видимости страницы. Если Visible равно False, то страница невидима и недоступна. Обратите внимание, что хотя бы одна страница на форме всегда должна быть видима.
- **Control** – специальное свойство, возвращающее элемент управления по имени. Можно также использовать для проверки существования элемента управления. Если элемента с заданным именем не существует, свойство просто возвращает Nothing.
- **ToolTip** – текст всплывающей подсказки, который появится на экране при подведении курсора мыши к ярлычку страницы в нижней части окна формы.
- **Index** – текущий номер в списке страниц.
- **PageSetup** – возвращает ссылку на объект PageSetup, связанный с управлением параметрами печатной страницы.
- **Copies** – устанавливает количество копий документа на странице для печати. При печати очередной копии возникает событие **OnPrintCopy**. Это событие можно использовать для установки значений в элементах управления (например номер экземпляра).

Объект `FrmPage` поддерживает следующий список методов.

- **`Activate`** – активизирует страницу. Родительская рабочая книга получит событие `OnActivate`.
- **`ElementAt`** – возвращает элемент управления, который находится на месте с заданными координатами.

В процессе своей работы объект `FrmPage` может генерировать обрабатывать следующие события.

Напомним, что хотя обработчики этих событий и находятся в модуле самой формы, префиксом для них будет имя страницы. Например, если страница имеет имя `Page1`, то обработчиком события `OnClick` для этой страницы будет процедура `Page1_OnClick`. Обратите внимание, что события элементов управления будут обрабатываться в том же самом контексте.

- **`OnActivate`** – возникает при активизации страницы.
- **`OnContextMenu`** – возникает при нажатии правой кнопки мыши на странице. Если щелкнуть на элементе управления, то возникнет событие для элемента, а не для страницы формы.
- **`OnPrintCopy`** – возникает при печати очередной копии документа (если свойство **`Copies`** больше единицы)

Общие свойства и методы для элементов

Имеется несколько свойств и методов, общих для всех элементов управления на форме:

- **`Name`** – имя элемента управления, под которым он известен в программе. Имя вполне может быть пустым. Если это так, то сослаться на элемент из программы невозможно. В процессе выполнения программы изменить значение имени невозможно.
- **`Left`**, **`Top`**, **`Height`**, **`Width`** – координаты и размеры элемента управления. Для форм они задаются в десятых долях миллиметра относительно верхнего левого угла печатной страницы (без учета полей). Установка этих свойств во время выполнения программы приводит к тому, что элемент либо перемещается, либо меняются его размеры.

- **Visible** – если свойство равно `True`, то элемент выводится на экран, в противном случае он скрыт.
- **TabStop** – если свойство равно `True`, то элемент включается в список перехода по клавишам Tab и Shift+Tab.
- **TabIndex** – позиция элемента в списке перехода по клавишам Tab и Shift+Tab.
- **Tag** – произвольная текстовая строка, невидимая пользователю. Может использоваться по усмотрению программиста.
- **ToolTip** – всплывающая подсказка, которая будет появляться при наведении курсора мыши на элемент.
- **Parent** – (только чтение) ссылка на родительский объект. В данном случае это будет страница формы (`FrmPage`).
- **TypeName** – (только чтение) текстовая строка, определяющая имя типа объекта.
- **Layout** – специальное свойство, определяющее как элемент будет печататься на форме, содержащей таблицу с прокруткой. Подробнее оно будет описано в разделе, посвященном работе с такими таблицами (`FrmScrollGrid`).

Все элементы управления формы имеют один общий метод:

- **SetFocus** – устанавливает фокус ввода на данный элемент управления. Если элемент не виден, то страница прокручивается так, чтобы он стал виден.

Далее будут перечислены свойства, применимые почти для всех элементов, кроме некоторых (это оговорено особо).

- **Enabled** – если элемент предназначен для взаимодействия с пользователем, например кнопка или редактор, то он обязательно имеет свойство `Enabled`. Если оно установлено в `False`, то элемент запрещен и не воспринимает действия пользователя. Если это возможно, то запрещенные элементы визуально выделяются серым цветом.
- **BackColor** – цвет фона для элемента управления это свойство неприменимо к линиям и таблицам (каждая ячейка или колонка таблицы может иметь свой цвет фона.
- **ForeColor** – цвет текста для элемента.

- **Font** – шрифт для элемента управления. Шрифт представлен стандартным специальным объектом Font. Его свойства и методы описаны в разделе, посвященном стандартным объектам *VBScript*.
- **Multiline** – если это свойство установлено в True, то текст элемента управления будет автоматически переноситься по словам.
- **PrintBorder** – если свойство равно True, то рамки, окружающие элемент управления будут выводиться на печать.
- **BorderStyle** и **BorderColor** – группа свойств, позволяющая избирательно управлять рамками вокруг элемента управления. Подробности см. в электронной справочной системе.
- **PrintText** – если свойство равно True, то текст элемента управления будет выводиться на печать.
- **PrintBack** – если свойство равно True, то фоновый цвет заливки элемента будет выводиться на печать.
- **ToolTip** – определяет текст всплывающей подсказки, которая будет выводиться в маленьком окне, появляющемся когда курсор мыши находится над элементом управления.
- **Tag** – произвольная текстовая строка, которую программист может использовать по своему усмотрению.

Если элемент управления поддерживает *связывание данных*, то у него имеется специальный метод Recalc, который пересчитывает значения данных.

Связывание данных

Многие элементы управления на форме могут быть *связаны* с переменными или свойствами. Связывание означает, что элемент будет «отслеживать» изменение значения связанной переменной или свойства и наоборот, изменения текста в элементе управления будет приводить к изменению значения переменной или свойства. Такой подход позволяет значительно сократить размер программы и упростить ее логику.

Для того чтобы задействовать механизм связывания, нужно установить значение свойства **DataSource**.

DataSource – основное свойство для связывания. Оно содержит текстовую строку, представляющую собой выражение *VBScript*, которое будет

вычисляться при связывании. Если элемент может взаимодействовать с пользователем (например, это редактор текста), то выражение должно быть доступно для установки значения. Другими словами, оно должно быть допустимым для левой части оператора присваивания. Элемент, у которого значение свойства **DataSource** не пустое, называется *связанным*.

Рассмотрим, как работает связывание. После загрузки формы, а так же при любых изменениях в связанных элементах управления, происходит *пересчет* формы. Пересчет заключается в следующем.

Для всех связанных элементов вычисляются выражения, указанные в `DataSource`. Полученные значения присваиваются элементу управления и отображаются на форме. Значение элемента управления чаще всего представлено свойством `Value`. Обратите внимание, что при пересчете формы связанные значения только читаются. Это означает, что если элемент не допускает ввода (например, текстовая метка `FrmLabel`), то в качестве `DataSource` можно указывать любое выражение *VBScript*, включая арифметические и логические операторы, вызовы встроенных функций, в общем, все что угодно.

При изменениях в элементах управления происходит следующее.

Сначала устанавливается соответствующее значение элемента управления (свойство `Value`). Если `DataSource` для элемента не пусто, то оно используется для записи получившегося значения. Фактически, выполняется такое выражение:

```
(element.DataSource) = element.Value
```

где под `(element.DataSource)` понимается реальное значение того, что написано в `DataSource`.

После этого возникает событие `OnBound` для связанного элемента управления. Это событие имеет аргумент `Lock`, передаваемый по ссылке. Если обработчик события не установил `Lock` в `True` (или обработчика нет вообще), то после этого запускается пересчет формы.

В самой форме имеется два специальных события, которые предназначены для более тонкого управления пересчетом.

- **`BeforeRecalc`** – это событие возникает перед пересчетом всей формы. Событие имеет аргумент `Cancel`, передаваемый по ссылке. Если обработчик события `BeforeRecalc` установит `Cancel` в `True`, то дальнейшего пересчета не будет.

- **AfterRecalc** – простое событие без аргументов, которое возникает после пересчета формы.

Для полной блокировки пересчета, включая возникновение событий BeforeRecalc и AfterRecalc, в форме имеется специальное свойство:

- **LockRecalc** – если оно установлено в True, то пересчет заблокирован и не выполняется.

Элементы управления для форм

Элемент FrmLabel

Элемент FrmLabel (текстовая метка) предназначен для размещения на форме не редактируемых текстовых полей. Метка может быть закрашена и может иметь рамки. Причем каждая сторона рамки независима от других сторон. Такой подход позволяет очень просто создавать сложные бланки документов. Важнейшие свойства метки таковы:

- **Caption** – текст, который будет выводиться внутри метки. Свойство Text просто дублирует Caption.
- **Multiline** – если равно True, то текст метки может располагаться в несколько строк.
- **Alignment** – выравнивание текста метки. Задается следующими константами:

acLeft	выравнивание по левому краю
acCenter	выравнивание по центру
acRight	выравнивание по правому краю

- **Vertical** – если равно True, то текст метки будет написан вертикально (снизу вверх).

Элемент FrmLabel может работать как гиперссылка. В этом случае при наведении курсора мыши на метку, он изменяет форму на стандартный указатель для ссылок (в виде руки), а метка подчеркивается и меняет цвет. При щелчке левой кнопкой мыши на такой метке возникает событие OnClick.

Для того чтобы сделать метку гиперссылкой, установите значение свойства HLink в True. Цвет активной ссылки задается с помощью свойства

ALinkColor. Обратите внимание, что если HLink установлено в False, событие OnClick не возникает.

Элемент FrmLine

Элемент FrmLine очень прост и представляет собой просто прямую линию на форме. Линия может быть сплошной, пунктирной, штриховой и так далее. Линия не содержит стандартных свойств TabIndex и TabStop, другими словами, пользователь не может установить на нее фокус ввода. Основные свойства линии FrmLine следующие:

- **X1,X2,Y1,Y2** – координаты концов линии на форме. Как и все координаты на форме задаются в 0.1 мм.
- **LineColor** – цвет линии.
- **LineStyle** – стиль (включая толщину) линии. Стиль задается константами, указанными ниже.
- **PrintLine** – если это свойство установлено в True, то линия печатается на принтере, в противном случае – нет.

Константы для стилей линий

Константа	Значение	Описание
acLineNone	0	Нет линии
acLineNormal	1	Тонкая линия
acLineMiddle	2	Линия средней толщины
acLineBold	3	Жирная линия
acLineDash	23	Пунктирная линия
acLineDot	24	Линия из точек
acLineDashDot	25	Штрих пунктирная линия
acLineDashDotDot	26	Штрих точка точка
acLineDouble	27	Двойная линия

Элемент FrmPicture

Элемент управления FrmPicture предназначен для размещения на форме различных изображений. Помимо общих свойств, он имеет следующие дополнительные свойства.

- **Center** – если свойство равно True, то картинка будет отцентрирована в пределах размеров элемента управления.

- **Stretch** – если свойство равно True, то картинка будет масштабироваться в соответствии с размерами элемента .
- **Picture** – возвращает ссылку на стандартный объект Picture, который и хранит картинку.
- **PrintPicture** – если это свойство установлено в True, то картинка печатается на принтере, в противном случае – нет.

Примечание. Внутри объекта хранится не ссылка на файл картинки, а сами данные картинки. Таким образом, в процессе выполнения программы, наличия файла картинки не требуется. Обычно картинка устанавливается в процессе конструирования формы. Если нужно менять картинку в процессе выполнения, то ее нужно сначала загрузить стандартной функцией LoadPicture, а затем присвоить полученную ссылку свойству Picture. В этом случае, разумеется, файл картинки должен быть доступен в процессе выполнения программы.

Элемент FrmButton

Элемент управления FrmButton это обычная командная кнопка. При ее нажатии на форме возникает событие OnClick. У кнопки имеется метод DoClick, вызов которого имитирует нажатие на кнопку.

Если фокус стоит на кнопке, то на кнопке выводится пунктирный прямоугольник. В этом случае нажатие клавиш Enter или Space приводит к нажатию кнопки.

Элемент FrmCheck

Элемент управления FrmCheck это кнопка-флажок. Основные свойства элемента следующие:

- **Checked** – отмечен ли флажок.
- **Flat** – способ отображения элемента. Если свойство равно True, то флажок отображается «плоским», если False, то с трехмерными эффектами.

При изменении состояния флажка возникает событие OnClick. У флажка имеется метод DoClick, вызов которого имитирует щелчок на флажке.

Элемент FrmRadio

Элемент управления FrmRadio это кнопка-переключатель. В отличие от флажка (FrmCheck), в один и тот же момент времени, только один переключатель из группы может быть отмечен. Однако, ответственность за это лежит на программисте.

Основные свойства элемента следующие:

- **Checked** – отмечен ли переключатель.

При изменении состояния переключателя возникает событие OnClick. У него также имеется метод DoClick, вызов которого имитирует щелчок на переключателе.

Элемент FrmRect

Элемент управления FrmRect представляет собой прямоугольник. Используется для визуального оформления форм. Поскольку он никак не взаимодействует с пользователем, то установить на него фокус ввода невозможно.

Элемент не имеет никаких специфических свойств и методов и не генерирует событий.

Элемент FrmEdit

Элемент FrmEdit (текстовый редактор) предназначен для ввода информации в виде текста, чисел или дат. Редактор может быть многострочным.

Можно вводить в редактор не только текст, но и числа или даты. В этом случае можно задать тип и автоматическое форматирование данных. Тип данных определяется значением свойства DataType. Формат отображения задается свойством Format.

Редактор поддерживает *связывание* данных. Основные свойства редактора следующие:

- **Alignment** – задает выравнивание данных.
- **DataSource** – источник данных для связывания.
- **DataType** – задает тип данных, которые хранятся в редакторе.
- **Format** – текстовая строка, определяющая формат отображения данных.

- **Text** – текст, который отображается в редакторе.
- **Value** – данные, которые хранятся в редакторе. Если DataType равно vbString (то есть в редакторе хранится текст), то свойство Value равно свойству Text.

Редактор генерирует следующие события.

- **OnValueChanged** – возникает после того, как пользователь изменил текст в редакторе и редактор потерял фокус ввода. Если редактор связан с данными, вместо этого события возникает событие OnBound.
- **OnBound** – возникает после того, как произошло связывание данных с элементом управления.

Элемент FrmSelector

Элемент FrmSelector (селектор) представляет собой обычный текстовый редактор, в правой части которого выводится кнопка с многоточием. Помимо свойств методов и событий, характерных для редактора (FrmEdit), селектор генерирует событие OnClick при нажатии на кнопку выбора.

Обратите внимание, что если селектор связан с данными (его свойство DataSource) не пусто, то вместо OnClick будет возникать событие OnBound.

Элемент FrmDate

Элемент FrmDate (селектор даты) похож на обычный селектор, но кнопка выглядит по-другому, и при ее нажатии на экране появляется календарь, в котором можно выбрать дату.

Его поведение полностью аналогично обычному редактору, за исключением того, что тип данных для селектора даты всегда будет Date.

Когда пользователь изменяет дату в селекторе, возникает событие OnDateChanged.

Обратите внимание, что если селектор даты связан с данными (его свойство DataSource) не пусто, то вместо OnDateChanged будет возникать событие OnBound.

Элемент FrmComboBox

Элемент `FrmComboBox` (комбинированный список) используется для выбора одной из нескольких альтернатив. Элементы списка разделяются символом вертикальной черты |.

Помимо стандартных свойств, этот элемент управления имеет пару дополнительных.

- **`ComboContents`** – текстовая строка, определяющая список альтернатив для выбора.
- **`ComboContents2`** – текстовая строка, определяющая список альтернатив для выбора с дополнительными параметрами. Если содержимое списка задается с помощью свойства `ComboContents2`, то свойство `Value` будет возвращать или устанавливать не порядковый номер в списке, а дополнительный параметр.
- **`Value`** – целое число, определяющее текущий выбор.

При изменении выбора в списке возникает событие `OnSelChanged`.

Элемент FrmGrid

Элемент `FrmGrid` (таблица) один из наиболее мощных элементов управления. Он представляет собой таблицу, содержащую прямоугольную матрицу ячеек, каждая из которых управляется независимо от других.

Каждая ячейка таблицы может быть окрашена в свой цвет, иметь разные варианты выравнивания текста, может печататься разным шрифтом и иметь индивидуальные рамки.

Поддерживаются следующие типы ячеек (в скобках приведены константы, описывающие типы ячеек):

- **Метка** (`acLabel`)
- **Редактор** (`acEditor`)
- **Селектор** (`acSelector`)
- **Дата** (`acDate`)
- **Кнопка** (`acButton`)
- **Флажок** (`acCheck`)
- **Переключатель** (`acRadio`)
- **Список** (`acCombo`)

Можно заметить, что эти типы дублируют другие элементы управления формы.

Доступ к ячейкам таблицы основывается на понятии *диапазона*. Диапазон – это просто прямоугольный участок ячеек. Диапазон может состоять, в том числе, и из одной ячейки.

Применение диапазонов имеет одну особенность. Когда Вы читаете значение какого-либо свойства диапазона, система возвращает значение, связанное с левой верхней ячейкой диапазона. В процессе записи, однако, нужное свойство присваивается всем ячейкам диапазона.

Диапазон ячеек таблицы `FrmGrid` представлен отдельным объектом `FrmGridRange`.

Основные свойства объекта `FrmGrid` следующие (не считая общих).

- **`BindRow,BindCol`** – адрес ячейки для, которой проводится вычисление при пересчете таблицы. Обычно используется в выражениях, указываемых в свойстве `DataSource`.
- **`DisplayGrid`**– отображать или нет сетку.
- **`Row,Column`** – адрес текущей (в которой находится курсор) ячейки.
- **`Rows,Columns`** – количество строк и столбцов в таблице.
- **`Cell`** – возвращает ссылку на одну ячейку.
- **`EntireRow,EntireColumn`** – возвращают ссылку на диапазон, равный целой строке или колонке.
- **`EntireGrid`** – возвращает ссылку на диапазон, включающий всю таблицу.
- **`Range`** –возвращают ссылку на заданный диапазон.
- **`PrintGrid`** – печатать или нет сетку таблицы.

Таблица `FrmGrid` поддерживает следующие методы.

- **`DeleteRow,DeleteColumn`** – удаляет строку или колонку.
- **`InsertRow,InsertColumn`** – вставляет новую строку или колонку.
- **`Recalc`** – пересчитывает таблицу.

- **SetPos** – устанавливает курсор в заданную позицию.

В процессе своей работы таблица FrmGrid может генерировать следующие события.

Напомним, что хотя обработчики этих событий и находятся в модуле формы, префиксом для них будет имя таблицы. Например, если таблица имеет имя Grid1, то обработчиком события OnCellClick для нее будет процедура Grid1_OnCellClick.

- **OnCellChanged** – возникает при любых изменениях содержимого ячеек. В процедуру обработки события передаются координаты измененной ячейки.
- **OnCellClick** – возникает при нажатии кнопки в ячейках типа кнопка, флажок, переключатель или селектор.
- **OnPosChanged** – возникает при перемещениях по таблице.

Объект FrmGridRange

Объект FrmGridRange представляет собой прямоугольный участок ячеек таблицы, называемый *диапазоном*.

Применение диапазонов имеет одну особенность. Когда значение какого-либо свойства диапазона читается, система возвращает значение, связанное с левой верхней ячейкой диапазона. В процессе записи, однако, нужное свойство присваивается всем ячейкам диапазона.

Основные свойства объекта FrmGridRange следующие.

- **Alignment** – выравнивание текста.
- **BackColor,ForeColor** – цвет фона и цвет текста.
- **CellType** – тип ячеек для диапазона.
- **CellDataType** – тип данных для ячеек диапазона.
- **ColumnWidth,RowHeight –** ширина колонок и высота строк.
- **ComboContents** – текстовая строка, определяющая список альтернатив для выбора. Имеет смысл только для ячеек типа ComboBox.
- **DataType** – тип данных, которые хранятся в ячейках диапазона.
- **DataSource** – источник данных для связывания.

- **Font** – шрифт. Представляет собой стандартный объект Font.
- **Format** – текстовая строка, определяющая формат отображения данных.
- **Height,Width** – высота и ширина диапазона (в ячейках).
- **MergeCells** – если это свойство равно True, то все ячейки диапазона объединяются в одну большую ячейку.
- **Multiline** – если это свойство равно True, то разрешен перенос текста по словам.
- **Left, Top** – левая и верхняя координаты диапазона (в ячейках).
- **Value** – значение, которое хранится в ячейке.

Основные методы, поддерживаемые объектом FrmGridRange.

- **Equalize** – выравнивает ширину и высоту ячеек по большей.
- **Clear** – очищает значения ячеек диапазона.
- **SetBorder** – устанавливает рамки для диапазона.

Элемент FrmScrollGrid

Очень часто возникает необходимость создания бланков документов, содержащих табличную часть переменной длины. Для этого используется элемент управления FrmScrollGrid (таблица с прокруткой).

В отличие от обычной таблицы FrmGrid, таблица с прокруткой помещается на экран не вся. В каждый момент времени видна только ее часть. Когда фокус ввода находится в таблице, справа от нее отображается стандартная полоса прокрутки. Ее можно использовать для навигации по ячейкам.

Когда документ, содержащий таблицу с прокруткой, печатается, то он разбивается на страницы в соответствии с реальным количеством строк в таблице. Причем другие элементы управления формы будут печататься на различных страницах в соответствии со значением их свойства Layout. Свойство Layout может иметь следующие значения:

- **(0) Nothing** (нигде)
- **(1) FirstPage** (только первая)
- **(2) EachPage** (каждая)
- **(3) LastPage** (только последняя)

Таким образом, все элементы имеющие значение свойства Layout равное 1 (FirstPage), будут печататься только на первой странице, имеющие значение Layout = 2 (EachPage) – на каждой странице, и, наконец, имеющие значение Layout = 3 (LastPage), будут печататься только на последней странице. Сама таблица с прокруткой должна иметь значение свойства Layout равное 0 (Nothing).

Поскольку таблица имеет переменное количество строк, то она организована не в виде матрицы ячеек, а в виде коллекции столбцов. Каждый столбец представлен объектом FrmSGridColumn. Столбец, в свою очередь, состоит из набора ячеек, каждая из которых представлена объектом FrmSGridCell.

Обратите внимание, что все ячейки, находящиеся в одной колонке имеют общие атрибуты, такие как цвет, шрифт, источник данных и т.д.

Как и в обычной таблице, в таблице с прокруткой поддерживаются следующие типы ячеек (в скобках приведены константы, описывающие типы):

- **Метка** (acLabel)
- **Редактор** (acEditor)
- **Селектор** (acSelector)
- **Дата** (acDate)
- **Кнопка** (acButton)
- **Флажок** (acCheck)
- **Переключатель** (acRadio)
- **Список** (acCombo)

Основные свойства объекта FrmScrollGrid следующие:

- **AutoGrow** – разрешает или запрещает автоматическое добавление строк при вводе данных в последнюю строку таблицы. При автоматическом добавлении строк не учитываются колонки, для которых свойство ExcludeAutoGrow установлено в True.
- **AutoHeight** – разрешает или запрещает автоматическое увеличение высоты строк при вводе в ячейку длинного текста. Для того, чтобы это работало, в соответствующей колонке должен быть разрешен перенос по словам (свойство Multiline должно быть равно True).

- **BindRow,BindCol** – адрес ячейки для, которой проводится вычисление при пересчете таблицы. Обычно используется в выражениях, указываемых в свойстве `DataSource`.
- **DisplayGrid**– отображать или нет сетку.
- **GridColor** – цвет сетки.
- **Row,Col** – адрес текущей (в которой находится курсор) ячейки.
- **Rows,Columns** – количество строк и столбцов в таблице.
- **Cell** – возвращает ссылку на одну ячейку (`FrmSGridCell`).
- **Column** – возвращает ссылку на одну колонку таблицы (объект `FrmSGridColumn`).
- **PrintGrid** – печатать или нет сетку таблицы.
- **RowHeight** – высота каждой строки таблицы. Если свойство `AutoHeight` равно `True`, то реальная высота строки будет кратна значению свойства `RowHeight`.
- **ScrollPos** – свойство позволяющее управлять положением движка полосы прокрутки.
- **AlinkColor** – цвет для меток – ссылок. При наведении курсора мыши на ссылку, текст будет отображаться заданным этим свойством цветом и подчеркиваться. При щелчке на ссылке возникает событие `OnCellNavigate`. Установить ссылку можно с помощью свойства `HLink` объекта `FrmSGridColumn`.

Таблица `FrmScrollGrid` поддерживает следующие методы.

- **Clear** – очищает все значения в таблице.
- **DeleteColumn** – удаляет колонку.
- **InsertColumn** – вставляет новую колонку.
- **Recalc** – пересчитывает таблицу.
- **SetPos** – устанавливает курсор в заданную позицию.
- **Pack** – удаляет пустые строки снизу таблицы.

- **PackAutoGrow** – удаляет пустые строки в нижней части таблицы с учетом значения свойства `ExcludeAutoGrow` (устанавливается для колонок таблицы.
- **ScrollDown** – устанавливает положение полосы вертикальной прокрутки таблицы так, чтобы была видна последняя строка таблицы.
- **EnableRanges** – разрешает работу с диапазонами ячеек таблицы. Если диапазоны разрешены, то пользователь может выделить строки, столбцы или прямоугольные диапазоны с помощью правой кнопки мыши.

В процессе своей работы таблица `FrmScrollGrid` может генерировать следующие события.

- **OnCellChanged** – возникает при любых изменениях содержимого ячеек. В процедуру обработки события передаются координаты измененной ячейки.
- **OnCellClick** – возникает при нажатии кнопки в ячейках типа кнопка, флажок, переключатель или селектор.
- **OnCellNavigate** – возникает при щелчке на ячейке - гиперссылке.
- **OnPosChanged** – возникает при перемещениях по таблице.
- **OnScroll –** возникает при прокрутке таблицы.
- **OnPrintPage –** возникает при печати одной из страниц формы, на которой находится таблица. Событие позволяет выполнить какие-то действия перед печатью страницы. Например, можно подсчитать и напечатать промежуточные итоги по странице.
- **OnRangeTrack, OnRangeSelect –** возникают в процессе выделения (`OnRangeTrack`) и по завершении выделения диапазона ячеек. Эти события работают только тогда, когда разрешена работа с диапазонами ячеек (метод `EnableRanges`).
- **OnCustomBind –** специальное событие, которое используется в процессе расширенного связывания.

Объект FrmSGridColumn

Объект FrmSGridColumn представляет собой один столбец таблицы с прокруткой (FrmScrollGrid).

Ссылка на объект может быть получена с помощью свойства Column таблицы FrmScrollGrid.

Все ячейки таблицы, которые входят в один столбец имеют одинаковые атрибуты цвета, шрифта, выравнивания и т.д. Единственное, чем они отличаются, так это значением.

Основные свойства объекта FrmSGridColumn следующие.

- **Alignment** – выравнивание текста.
- **BackColor,ForeColor** – цвет фона и цвет текста.
- **CellType** – тип ячеек для колонки.
- **CellDataType** – тип данных для ячеек колонки.
- **ColumnWidth** – ширина колонки.
- **ComboContents** – текстовая строка, определяющая список альтернатив для выбора. Имеет смысл только для ячеек типа ComboBox.
- **DataSource** – источник данных для связывания.
- **Font** – шрифт. Представляет собой стандартный объект Font.
- **Format** – текстовая строка, определяющая формат отображения данных.
- **Multiline** – если это свойство равно True, то разрешен перенос текста по словам.
- **ExcludeAutoGrow –** позволяет исключить колонку из рассмотрения при работе механизма автоматического добавления строк.
- **HLink –** устанавливает признак, разрешающий ячейкам таблицы выглядеть и работать как гиперссылки.
- **CustomBind –** включает механизм расширенного связывания для колонки.

Объект FrmSGridCell

Объект `FrmSGridCell` представляет собой одну ячейку таблицы. Объект очень прост и имеет только несколько свойств.

- **`Value`** – значение, которое хранится в ячейке таблицы.
- **`Parent`** – ссылка на родительский столбец `FrmSGridColumn`.
- **`Grid`** – ссылка на родительскую таблицу `FrmScrollGrid`.

Объект `FrmSGridCell` имеет всего один метод.

- **`SetBkColor`** – устанавливает цвет фона ячейки независимо от цвета фона, установленного для колонки.

Глава VIII.

Диалоги

В этой главе описана работа с диалогами *Акцент* и объектная модель диалогов. Приводится полное описание всех элементов управления, которые могут находиться на панели диалога, их свойств, методов и событий.

Диалог в программах семейства *Акцент* представляет собой обычное диалоговое окно *Windows*. Диалоги используются для ввода информации пользователем. Основное отличие диалогов от форм то, что диалог нельзя напечатать (по крайней мере, стандартными средствами). Кроме того, в отличие от форм, все диалоги в программе *модальные*. Это означает, что перед тем, как продолжить работу с программой, пользователь обязан завершить работу с диалогом. Однако диалоги вполне могут быть вложенными, то есть можно из одного диалога вызвать другой.

На панели диалога располагаются *элементы управления*. Они похожи на элементы управления для форм, но ведут себя несколько по-другому. Имеются некоторые специфические для диалогов элементы управления, например закладки и таблицы. Как и формы, диалоги поддерживают механизм *связывания* (*binding*).

Помимо простых элементов управления, таких как текстовые редакторы или кнопки, в диалоге могут использоваться достаточно сложные элементы, характерные для современных версий *Windows*. Это таблицы, сложные списки, селекторы дат, «деревья» и многое другое.

Объектная модель для диалогов

На рисунке 4 приведена объектная модель для диалогов. Она включает всего два уровня иерархии. На верхнем уровне находится сам диалог, а на следующем – элементы управления диалога.

Диалог включает программу, которая обрабатывает события как от самого диалога, так и от его элементов управления.

Ошибка! Ошибка связи.

Рис. 4. Объектная модель для диалогов

Создание диалога

В программах семейства *Акцент* создание диалога возможно следующим способом. Объект диалога создается вызовом метода `CreateDialog` объекта `Application`.

Метод `CreateDialog` имеет следующий синтаксис.

Синтаксис

```
Set d = app.CreateDialog(Dlg,[OpID],[Fld],[Tml])
```

Аргументы

`Dlg` – Обязательный аргумент типа `Variant`. Если это целое число, то оно рассматривается как идентификатор диалога, подключенного к проекту текущей базы данных. Если это символьная строка, то она рассматривается как имя файла диалога `ADG`. Если путь к файлу не указан, учитывается информация о расположении файлов из файла расположения `ARD`.

`OpID` – Необязательно. Идентификатор операции, которая будет загружена в диалог. Ссылку на эту операцию можно получить с помощью свойства `CurrentOperation`. Если этот аргумент не указан, `CurrentOperation` вернет ссылку на новую пустую операцию.

`Fld,Tml` – Необязательно. Идентификаторы папки и шаблона для загруженной в диалог операции. Используются только для вновь созданных операций.

Не забывайте о том, что метод возвращает ссылку на объект, поэтому использование ключевого слова `Set` является обязательным.

Обратите внимание на то, что созданный методом `CreateDialog` диалог только загружен в память компьютера. Чтобы увидеть его на экране, нужно выполнить метод `DoModal` самого диалога.

Размеры диалога

Все размеры внутри диалога, включая размеры и положение элементов управления, задаются в специальных единицах, называемых `DLU` (*Dialog Units*). Коэффициент пересчета `DLU` в пиксели экрана зависит от выбранного шрифта. Такой подход позволяет получить правильные размеры са-

мого диалога и элементов управления на нем независимо от установленного разрешения экрана и от установленного для диалога шрифта.

В отличие от форм, в которых каждый элемент может выводиться своим шрифтом, все элементы управления диалога отображаются одним и тем же шрифтом. Этот шрифт определяется следующими свойствами:

- **FontName** – определяет гарнитуру (наименование) шрифта для диалога.
- **FontSize** – определяет размер шрифта для диалога. Обратите внимание, что размер шрифта задается в пунктах (1 пункт равен $^1/_{72}$ дюйма), а не в DLU. Так сделано потому, что конкретное значение DLU зависит именно от размера шрифта.

Обычно для диалогов устанавливается шрифт *MS Sans Serif* размером 8 пунктов.

Диалоги в программах семейства *Акцент* поддерживают возможность изменения размеров в процессе работы. Для того чтобы разрешить эту возможность, нужно установить свойство Resizable в значение True. После этого появится возможность изменять размеры диалога стандартными для *Windows* средствами.

Можно «привязывать» элементы управления к границам диалога. Этим управляет специальное свойство Snap, которое есть у всех элементов управления.

Если нужно, чтобы диалог сохранял свои размеры между сеансами, то нужно установить свойство SnapName. Этому свойству присваивается уникальная текстовая строка. Фактически это имя ключа в реестре, в котором будет сохраняться информация о текущих размерах диалога. Если свойство Resizable равно False, то значение SnapName не используется.

Объект Dialog

Объект Dialog находится на вершине иерархии и представляет собой само окно диалога. Он содержит в себе все элементы управления. Кроме этого, в диалоге есть карта Map. Ссылка на нее возвращается одноименным свойством Map. Более подробно работа с картой описана на стр. 22.

Основные свойства объекта Dialog следующие.

- **Caption** – текстовая строка, определяющая заголовок окна диалога.

- **Control** – обеспечивает доступ к элементу управления по его имени.
- **CurrentOperation** – возвращает ссылку на текущую операцию (объект Operation), связанную с формой. Если никакая операция не была связана с диалогом, то будет создана новая операция.
- **LockRecalc** – если это свойство имеет значение True, то пересчет диалога не выполняется и события BeforeRecalc и AfterRecalc не возникают.
- **Map** – возвращает ссылку на карту Map.
- **FileName** – имя файла текущего диалога (ADG) без указания пути.
- **PathName** – полное имя файла (включая путь) диалога (ADG).
- **Height,Width** – высота и ширина окна диалога в DLU.
- Динамические свойства только для чтения, имена которых совпадают с именами элементов управления просто возвращают ссылки на соответствующие элементы.

Для выполнения действий используются следующие методы.

- **DoModal** – выполняет диалог. Обычно вызывается из внешнего по отношению к диалогу контекста.
- **Unload** – выгружает диалог из памяти. Получает в качестве аргумента целое число, которое будет возвращено методом DoModal во внешний мир.
- **Recalc** – пересчитывает диалог.
- **ResetFocus** – убирает фокус ввода со всех элементов управления и устанавливает его на сам диалог.
- **ShowPopup** – позволяет показать на экране всплывающее меню. При выборе команды из такого меню генерируется событие OnPopup.
- **FireUserEvent** – вызывает пользовательское событие OnUser. Метод можно вызывать из другого контекста, что, в частности, позволяет организовать взаимодействие нескольких объектов системы в процессе выполнения программы.

В процессе работы объект Dialog может получать и обрабатывать следующие события.

- **BeforeRecalc,AfterRecalc** – возникают перед/после пересчета диалога. Позволяют контролировать процесс пересчета. В частности, событие BeforeRecalc имеет аргумент Cancel. Если установить его в True, то пересчет выполняться не будет.
- **CanClose** – возникает при попытке закрыть диалог. Имеется возможность запретить закрытие либо, выполнив дополнительные действия, разрешить. Обратите внимание, что событие CanClose вызывается всегда, независимо от способа, которым закрывается диалог.
- **OnPopup** – возникает при выборе команды из всплывающего меню. Такое меню может быть вызвано методом ShowPopup самого диалога или с помощью специального вспомогательного объекта PopupMenu (стр. 133).
- **OnLoad** – возникает после загрузки диалога, но до появления его на экране. Обычно в обработчике этого события выполняются все действия по инициализации элементов управления диалога.
- **OnHelp** – событие возникает когда пользователь желает получить справку о работе с диалогом. Это происходит при нажатии клавиши F1 на клавиатуре или при выборе команды **«Что это значит?»** из всплывающего меню. Для вызова справочной системы используется метод WinHelp специального объекта WinAPI (стр. 136).
- **OnFnKey** – событие возникает при нажатии функциональных клавиш (F1-F12). Событие возникает всегда, независимо от того, какой элемент управления находится в фокусе ввода.

Элементы управления диалогов

На панели диалога располагаются элементы управления. При взаимодействиях с пользователем эти элементы могут генерировать различные события. Следует отметить, что все обработчики событий находятся в контексте самого диалога и именуются стандартным способом в виде:

```
имяЭлемента_имяСобытия
```

Например, при нажатии кнопки с именем cmdButton система будет пытаться выполнить процедуру cmdButton_OnClick. Отсюда автомати-

чески следует, что для того, чтобы можно было обрабатывать события от элемента управления, нужно чтобы элемент имел непустое имя.

Возможные типы элементов управления приведены в таблице ниже.

Элемент	Назначение
`DlgLabel`	Текстовая метка (не редактируемая)
`DlgFrame`	Рамка
`DlgGroupBox`	Рамка группы
`DlgPicture`	Картинка
`DlgButton`	Командная кнопка
`DlgCheck`	Флажок
`DlgRadio`	Переключатель
`DlgEdit`	Текстовый редактор
`DlgSelector`	Текстовый редактор с кнопкой для выбора
`DlgDate`	Элемент управления для ввода даты
`DlgListBox`	Список
`DlgComboBox`	Комбинированный список для выбора
`DlgMeter`	Индикатор прогресса
`DlgGrid`	Таблица
`DlgTabStrip`	Закладки
`DlgListView`	Сложный список
`DlgTreeView`	Дерево

Общие свойства и методы для элементов

Имеется несколько свойств и методов, общих для всех элементов управления диалога:

- **`Name`** – имя элемента управления, под которым он известен в программе. Имя вполне может быть пустым. Если это так, то сослаться на элемент из программы невозможно. В процессе выполнения программы изменить значение имени невозможно. Когда диалог начинает выполняться, то в него добавляются динамические свойства только для чтения, совпадающие с именами элементов управления. Такие свойства просто возвращают ссылку на соответствующий элемент. Это позволяет в программе диалога обращаться к элементам просто по именам.
- **`Left`**, **`Top`**, **`Height`**, **`Width`** – координаты и размеры элемента управления. Для диалогов они задаются в `DLU` относительно верхнего левого угла диалога. Установка этих свойств во время выполне-

ния программы приводит к тому, что элемент либо перемещается, либо изменяются его размеры.

- **Visible** – если свойство равно `True`, то элемент выводится на экран, в противном случае он скрыт. Обратите внимание, что при использовании элемента управления «Закладки» (`DlgTabStrip`), это свойство совместно со свойством `Enabled` используется для переключения закладок.
- **TabStop** – если свойство равно `True`, то элемент включается в список перехода по клавишам Tab и Shift+Tab.
- **TabIndex** – позиция элемента в списке перехода по клавишам Tab и Shift+Tab.
- **ToolTip** – всплывающая подсказка, которая будет появляться при наведении курсора мыши на элемент.
- **Tag** – произвольная текстовая строка, которую программист может использовать по своему усмотрению.
- **ParentTab** – текстовая строка, задающая имя родительской закладки в элементе `DlgTabStrip`.

Все элементы управления диалога имеют один общий метод:

- **SetFocus** – устанавливает фокус ввода на данный элемент управления. Именно тот элемент, на который установлен фокус ввода, будет реагировать на нажатие клавиш клавиатуры. Обычно такой элемент выделяется визуально (например пунктирной рамкой). Чаще всего фокус переключается с помощью клавиш Tab и Shift + Tab.

Далее будут перечислены свойства, применимые почти для всех элементов. Чтобы узнать, применимо ли свойство к конкретному элементу, обратитесь к электронной справочной системе.

- **Enabled** – если элемент предназначен для взаимодействия с пользователем, например кнопка или редактор, то он обязательно имеет свойство `Enabled`. Если оно установлено в `False`, то элемент запрещен и не воспринимает действия пользователя. Запрещенные элементы визуально выделяются серым цветом.
- **Multiline** – если это свойство установлено в `True`, то текст элемента управления будет автоматически переноситься по словам.

- **`Caption`** – текст, отображаемый на элементе управления. Чтобы добавить к элементу «горячую клавишу», используемую для быстрого доступа, используется символ `&`. Поместите его перед нужной буквой и для перехода к элементу можно будет использовать комбинацию клавиш Alt+Буква. Обратите внимание, что если на элемент нельзя установить фокус (например метка `DlgLabel`), то фокус установится на следующий по порядку элемент. Это позволяет, в частности, организовать быстрый доступ к редакторам или спискам.
- **`Snap`** – набор битовых флажков, которые управляют «привязкой» элемента управления к границам диалога с изменяемыми размерами. Для того чтобы разрешить изменение размеров диалога, установите в `True` значение свойства `Resizable` самого диалога. Возможные значения флажков приведены ниже.

Значения флажков для свойства Snap

Значение	Описание
1	Привязка левой границы
2	Привязка верхней границы
4	Привязка правой границы
8	Привязка нижней границы

Например, если значение `Snap` установлено в `1 + 4 = 5`, то при изменении ширины диалога, будет синхронно изменяться ширина элемента, а если `Snap` равен 4, то элемент будет «приклеен» к правой границе и не будет изменяться в размерах.

Если элемент управления поддерживает *связывание данных*, то у него имеется свойство `DataSource` и специальный метод `Recalc`, который пересчитывает значения данных. Более подробно связывание данных описано на стр. 178.

Элемент DlgLabel

Элемент `DlgLabel` (метка диалога) представляет собой обычную текстовую метку, используемую для вывода нередактируемых подписей. Метка может быть как однострочной (`Multiline = False`), так и многострочной (`Multiline = True`). Метки поддерживают связывание данных, разумеется только для чтения. Помимо общих, метка имеет несколько дополнительных свойств.

- **Autosize** – если это свойство равно True, то размеры метки автоматически подбираются так, чтобы вместить весь помещаемый в нее текст.
- **Alignment** – выравнивание текста метки. Задается следующими константами:

acLeft	выравнивание по левому краю
acCenter	выравнивание по центру
acRight	выравнивание по правому краю

- **Sunken** – если свойство равно True, то метка будет обрамлена трехмерной рамкой.

Элемент DlgFrame

Элемент DlgFrame представляет собой обычную прямоугольную рамку, которая используется для оформления диалогов. Рамка может быть как просто черной, так и отображаться с трехмерными эффектами. Если свойство Etched установлено в True, то рамка отображается с 3-D эффектами. Никаких дополнительных особых свойств или методов рамка DlgFrame не поддерживает.

Элемент DlgGroupBox

Элемент DlgGroupBox похож на рамку DlgFrame, но дополнительно имеет заголовок в верхней левой части. Заголовок задается с помощью свойства Caption.

Элемент DlgGroupBox обычно используется для группировки зависимых переключателей DlgRadio. Все переключатели, которые размещены в пределах одной группы DlgGroupBox, будут зависеть друг от друга.

Элемент DlgPicture

Элемент управления DlgPicture предназначен для размещения на панели диалога различных изображений. Для работы с изображениями можно использовать файлы **bmp, gif, jpg, ico** и некоторые другие.

Помимо общих свойств, объект имеет следующие дополнительные свойства.

- **Sunken** – если свойство равно True, то картинка будет обрамлена трехмерной рамкой.

- **Picture** – возвращает ссылку на стандартный объект Picture, который и хранит картинку.

Примечание. Внутри объекта хранится не ссылка на файл картинки, а сами данные картинки. Таким образом, в процессе выполнения программы сам файл картинки не нужен. Обычно картинка устанавливается в процессе конструирования диалога. Если нужно менять картинку в процессе выполнения, то ее нужно сначала загрузить стандартной функцией LoadPicture, а затем присвоить полученную ссылку свойству Picture. В этом случае, разумеется, файл картинки должен быть доступен в момент выполнения программы.

Элемент DlgButton

Элемент DlgButton представляет собой обычную командную кнопку. Помимо основных свойств, командные кнопки на панели диалога имеют две дополнительные особенности.

Одну из кнопок диалога можно пометить как кнопку «По умолчанию». Это означает, что если пользователь нажмет клавишу Enter, то сработает именно эта кнопка. Кнопка «по умолчанию» задается с помощью свойства Default, которое нужно установить в True. Поскольку только одна кнопка может быть помечена как Default, то при установке этого свойства Default для всех других кнопок сбрасывается в False.

Точно так же можно назначить одну из кнопок диалога кнопкой «Отмена». Такая кнопка будет автоматически срабатывать при нажатии на клавиатуре клавиши Esc. Чтобы сделать кнопку кнопкой «Отмена», установите свойство Cancel в значение True. Как и для Default это сбросит Cancel для всех остальных кнопок диалога.

Кнопка «По умолчанию» визуально выделяется на диалоге более жирной рамкой. Для кнопки «Отмена» никаких визуальных отличий нет.

При нажатии кнопки генерируется стандартное событие OnClick. Никаких других события кнопка не вызывает.

Кроме стандартного метода SetFocus кнопка поддерживает еще и метод DoClick, который позволяет программно «нажать» кнопку.

Элемент DlgCheck

Элемент `DlgCheck` это обычный флажок, имеющий два состояния «отмечен» и «не отмечен». Когда пользователь щелкает на элементе, его состояние меняется на противоположное.

Свойство `Checked` возвращает или устанавливает значение флажка. Если свойство равно `True`, флажок отмечен, если `False` – сброшен.

При изменении состояния флажка возникает стандартное событие `OnClick`.

Флажок имеет метод `DoClick`, позволяющий программно «щелкнуть» флажок.

Элемент DlgRadio

В отличие от флажка, элемент `DlgRadio` представляет собой зависимый переключатель. Только один из переключателей, входящих в группу, может быть выбран. Для группировки переключателей используется панель `DlgGroupBox`. Все переключатели, расположенные в пределах одной панели, зависят друг от друга.

Состояние переключателя определяется свойством `Checked`. Если оно равно `True`, то переключатель выбран, если `False`, то нет.

Когда пользователь щелкает по переключателю, возникает стандартное событие `OnClick`.

Флажок имеет метод `DoClick`, позволяющий программно «щелкнуть» флажок.

Элемент DlgEdit

Элемент `DlgEdit` представляет собой обычный текстовый редактор. Помимо стандартных, редактор имеет следующие свойства.

- **`Alignment`** – выравнивание текста редактора. Задается следующими константами:
  - **`acLeft`** выравнивание по левому краю
  - **`acCenter`** выравнивание по центру
  - **`acRight`** выравнивание по правому краю
- **`Multiline`** – если свойство равно True, редактор многострочный, в противном случае – однострочный.

- **Password** – признак того, что редактор используется для ввода пароля. В этом случае вводимый текст заменяется звездочками.
- **Text** – текст, содержащийся в редакторе.

Редактор поддерживает методы для работы с буфером обмена Windows.

- **Cut** – Вырезает выделенный текст и помещает его в буфер обмена.
- **Copy** – Помещает копию выделенного текста в буфер обмена.
- **Paste** – Вставляет содержимое буфера обмена в редактор, замещая текущий выделенный текст.

Редактор может использоваться для отображения не только текста, но и значений других типов. Для этого служат следующие свойства.

- **DataType** – задает тип данных, которые хранятся в редакторе.
- **Format** – текстовая строка, определяющая формат отображения данных.
- **Value** – данные, которые хранятся в редакторе. Если `DataType` равно `vbString` (то есть в редакторе хранится текст), то свойство `Value` равно свойству `Text`.

Редактор генерирует следующие события.

- **OnTextChanged** – возникает после того, как пользователь изменил текст в редакторе и редактор потерял фокус ввода. Если редактор связан с данными (`DataSource` не пусто), вместо этого события возникает событие `OnBound`.
- **OnChanged** – возникает после любого изменения текста в редакторе. В момент возникновения события, свойство `Text` содержит уже измененное значение.
- **OnBound** – возникает после того, как было выполнено связывание элемента управления с данными. Если свойство `DataSource` не установлено, то событие не возникает.

Элемент DlgSelector

Элемент `DlgSelector` состоит из однострочного текстового редактора и кнопки выбора в правой части поля редактирования. Селектор полностью аналогичен редактору `DlgEdit` и дополнительно, вырабатывает стандартное событие `OnClick` при нажатии кнопки выбора.

Элемент DlgDate

Элемент управления DlgDate представляет собой селектор даты. Он состоит из поля редактирования с кнопкой выбора. При нажатии кнопки выбора на экране появляется стандартный календарь для выбора даты.

Элемент DlgDate поддерживает связывание данных. Соответственно у него имеется свойство DataSource, метод Recalc и событие OnBound.

Основные свойства, которые поддерживает элемент DlgDate следующие.

- **Date** – дата, выбранная в селекторе. Тип данных для этого свойства всегда будет Date. Если установить пустую дату или значение Null, то селектор будет отображаться пустым.
- **Value** – полностью дублирует значение свойства Date.

При изменении содержимого селектора (если, конечно, не установлено DataSource), возникает событие OnDateChanged.

Элемент DlgListBox

Элемент DlgListBox представляет собой простой стандартный список. Элементом списка является текстовая строка. Пользователь имеет возможность выбрать один из элементов списка.

Если в списке больше элементов, чем можно показать, то справа появляется полоса прокрутки.

Основные свойства, поддерживаемые элементом DlgListBox, следующие.

- **Count** – Свойство только для чтения. Возвращает количество элементов в списке.
- **Sorted** – Если свойство равно True, то элементы списка будут отсортированы по алфавиту.
- **SelectedItem** – возвращает или устанавливает индекс (номер) выделенного в списке элемента. Если никакой элемент не выделен, свойство будет равно –1. Установка свойства в –1 приводит к тому, что выделение снимается со всех элементов списка.

- **ItemText** – Свойство только для чтения. Возвращает текст элемента списка по индексу. Индекс задается аргументом свойства и может изменяться от 1 до `Count`.
- **ItemParam** – Возвращает или устанавливает целое значение, которое запоминается вместе с элементом списка, но не отображается. Индекс элемента задается аргументом свойства и может изменяться от 1 до `Count`.

Список `DlgListBox` поддерживает следующие методы.

- **AddItem** – Добавляет строку в список. Метод получает строку в качестве аргумента и возвращает индекс добавленного элемента. Обратите внимание, что если список отсортирован, то индекс вновь добавленного элемента вполне может быть где-то посередине.
- **RemoveItem** – удаляет элемент из списка. Удаляемый элемент задается индексом.
- **Clear** – Очищает список (удаляет все элементы).

Когда пользователь изменяет выделенный в списке элемент, генерируется событие `OnSelChanged`. Кроме того, список генерирует стандартное событие `OnDblClick` в случае, если пользователь дважды щелкнул на списке левой кнопкой мыши.

Элемент DlgComboBox

Элемент `DlgComboBox` похож на список `DlgListBox` и отличается от него тем, что в исходном состоянии весь список не виден, а виден только выбранный элемент списка. Чтобы открыть список, нужно нажать кнопку в правой части элемента управления.

Свойства, методы и события комбинированного списка `DlgComboBox` полностью аналогичны обычному списку `DlgListBox`. Кроме того, `DlgComboBox` поддерживает дополнительный метод `ShowDropDown`, который позволяет программно открыть выпадающий список.

Элемент DlgMeter

Объект `DlgMeter` представляет собой обычный индикатор прогресса, который располагается в диалоге. Обратите внимание, что использовать стандартный индикатор `Meter` в диалоге невозможно, так как в диалоге нет строки состояния и индикатор будет просто не виден пользователю.

Элемент DlgMeter очень прост и имеет всего несколько свойств и методов.

- **Pos** – текущая позиция индикатора.
- **Min,Max** – минимальное и максимальное положение позиции индикатора.
- **Vertical** – если значение свойства равно True, то индикатор расположен вертикально, в противном случае – горизонтально.
- **BarColor** – задает цвет полоски индикатора.
- **Smooth** – если значение свойства равно True, то индикатор заполняется сплошной полоской, если False – индикатор заполняется дискретно («кубиками»). В режиме выполнения программы свойство только для чтения.
- **StepIt** – наращивает значение индикатора на один шаг.
- **SetStep** – устанавливает шаг изменения позиции индикатора.
- **OffsetPos** – наращивает позицию индикатора на заданное значение.

Элемент DlgGrid

Объект DlgGrid является одним из самых мощных и сложных элементов управления. Он представляет собой таблицу с возможностью редактирования содержимого ячеек.

Все ячейки таблицы независимы друг от друга. Для каждой ячейки задается тип ячейки, тип данных для ячейки, цвет фона, цвет текста, выравнивание, формат отображения данных

Таблица поддерживает следующие типы ячеек:

Вид	Описание
Метка	может показывать текст, число или дату
Редактор	может использоваться для ввода текста, чисел и дат
Селектор	то же, что и редактор, но с кнопкой выбора в правой части
Дата	редактор для ввода дат с возможностью выбора из выпадающего календаря
Кнопка	обычная командная кнопка

Флажок	флажок, похожий на `DlgCheckBox`
Переключатель	переключатель, похожий на `DlgRadioButton`
Комбинированный список	может использоваться для выбора значения из списка альтернатив

Ячейки таблицы могут быть связаны с данными. Более подробная информация о связывании приведена на стр. 178.

Основные свойства, поддерживаемые таблицей `DlgGrid`, следующие.

- **Rows**, **Columns** – количество строк и столбцов в таблице.
- **Subrows** – количество подстрок в каждой строке таблице.
- **FixColumns** – количество столбцов слева, защищенных от прокрутки.
- **RowHeight** – высота строки. Обратите внимание, что, в отличие от колонок, все строки таблицы имеют одинаковую высоту.
- **DisplayGrid** – показывать ли сетку разметки.
- **DisplayLeftHeader**, **DisplayTopHeader** – отображать ли заголовки слева и вверху таблицы.
- **DisplayRowNo** – отображать ли номера строк в заголовках строк слева.
- **ColumnCaption** – возвращает или устанавливает заголовок колонки для таблицы.
- **ColumnWidth** – возвращает или устанавливает ширину заданной колонки таблицы.
- **EmptyString** – возвращает или устанавливает текст, который отображается вместо пустой таблицы.
- **Cell**, **Range**, **EntireColumn**, **EntireRow**, **EntireGrid** – возвращают заданный диапазон ячеек таблицы.
- **Col**, **Row** – свойства только для чтения. Возвращают положение курсора в таблице (текущую колонку и строку).
- **BindCol**, **BindRow** – свойства только для чтения. Возвращают номер колонки и номер строки, которая связывается (Binding) в текущий момент. Обычно эти свойства используются в выражениях для источников данных `DataSource`.

Для того чтобы программно установить курсор на нужную ячейку используется специальный метод **SetPos**.

Таблица **DlgGrid** поддерживает методы для работы со строками/колонками:

- **InsertColumn**, **InsertRow** – вставляют в таблицу новую колонку или строку.
- **DeleteColumn**, **DeleteRow** – удаляют заданную колонку или строку.

Как и все элементы, поддерживающие связывание данных, таблица имеет метод **Recalc**.

В процессе работы таблица генерирует следующие события.

- **OnCellChanged** – возникает при любых изменениях содержимого таблицы. Обратите внимание, что если изменяемая ячейка связана с данными (`DataSource` не пустое), то вместо этого события возникает событие **OnBound**.
- **OnPosChanged** – возникает при изменении текущего положения курсора в таблице.
- **OnCellClick** – возникает при нажатии кнопки в ячейках типа `acSelector` и `acButton`.
- **OnRClick** – возникает при нажатии на таблице правой кнопки мыши. Обычно используется для вывода контекстных меню.

Объект DlgGridRange

Объект `DlgGridRange` представляет собой прямоугольный диапазон ячеек в таблице диалога `DlgGrid`. Обратите внимание, даже одна ячейка таблицы все равно будет представлена объектом типа `DlgGridRange`.

Применение диапазонов имеет одну специфическую особенность. При чтении любого свойств, возвращается значение, связанное с левой верхней ячейкой диапазона. При записи – устанавливаются свойства для всех ячеек, входящих в диапазон.

Получить ссылку на объект **DlgGridRange** можно с помощью следующих свойств (они относятся к объекту `DlgGrid`).

- **Cell** – возвращает ссылку на одну ячейку.

- **Range** – возвращает ссылку на прямоугольный диапазон ячеек.
- **EntireRow** – возвращает ссылку на диапазон, представляющий собой целую строку таблицы.
- **EntireColumn** – возвращает ссылку на диапазон, представляющий собой целый столбец таблицы.
- **EntireGrid** – возвращает ссылку на диапазон, представляющий собой всю таблицу.

Основные свойства, которые поддерживает объект `DlgGridRange`, следующие.

- **Alignment** – возвращает или устанавливает тип выравнивания текста для ячеек диапазона. Задается следующими константами:
  - **acLeft** выравнивание по левому краю;
  - **acCenter** выравнивание по центру;
  - **acRight** выравнивание по правому краю.
- **BackColor** – возвращает или устанавливает цвет фона для ячеек диапазона.
- **ForeColor** – возвращает или устанавливает цвет текста для ячеек диапазона.
- **CellType** – возвращает или устанавливает тип ячеек для диапазона. Возможные типы задаются следующими константами:
  - **acLabel** текстовая метка;
  - **acEditor** текстовый редактор;
  - **acSelector** селектор (редактор с кнопкой);
  - **acDate** селектор даты;
  - **acButton** командная кнопка;
  - **acCheck** флажок;
  - **acRadio** переключатель;
  - **acCombo** комбинированный список.
- **CellDataType** – возвращает или устанавливает тип данных для ячеек. Задается следующими константами:
  - **vbLong** целое число;
  - **vbBoolean** булево значение;
  - **vbString** текстовая строка;
  - **vbDouble** число с плавающей точкой;
  - **vbCurrency** денежное значение;

vbDate дата/время.

- **Value** – возвращает или устанавливает значение ячеек диапазона. Тип данных зависит от свойств `CellType` и `CellDataType`.
- **Format** – формат отображения данных для ячеек, содержащих суммы, числа и даты.
- **ComboContents** – возвращает или устанавливает содержимое комбинированного списка для ячеек типа `acCombo`. Свойство имеет тип `String`. Элементы списка разделяются символом вертикальной черты.
- **CellParam** – возвращает или устанавливает произвольный числовой (типа `Long`) параметр, связанный с каждой ячейкой диапазона.
- **DataSource** – возвращает или устанавливает источник данных для ячеек диапазона. Источник данных представляет собой выражение языка VBScript, которое используется для связывания. Более подробная информация о связывании приведена на стр. 178.

Объект поддерживает один метод **SetAttr**, который позволяет установить несколько свойств за один вызов.

Метод `SetAttr` имеет следующий синтаксис.

Синтаксис

SetAttr(Type,[DataType],[Align],[Back],[Fore])

Аргументы

`Type` – тип ячеек
`DataType` – тип данных для ячеек
`Align` – выравнивание для ячеек
`Back` – цвет фона для ячеек
`Fore` – цвет текста для ячеек

Элемент DlgTabStrip

Элемент управления `DlgTabStrip` предназначен для построения диалогов, использующих страницы свойств. Сам элемент представляет собой набор закладок для страниц свойств.

Имеется способ автоматического связывания элементов управления с закладками. Для этого используется свойство `ParentTab`, которое есть у

всех элементов диалога. Для того чтобы связать элемент с закладкой, установите свойство ParentTab для элемента в следующее значение:

```
<имя_элемента>:<имя_закладки>
```

Например, если нужно, чтобы кнопка находилась на первой закладке элемента DlgTabStrip с именем tab1, установите значение свойства ParentTab в "tab1:1".

Если элемент связан с закладкой, то при активизации закладки он делается видимым (Visible = True) и разрешенным (Enabled = True). Когда закладка становится неактивной, то элемент автоматически запрещается (Enabled = False) и делается невидимым (Visible = False).

Если закладка уже имеет имя, то при размещении элемента на такой закладке свойство ParentTab устанавливается автоматически.

Основные свойства, поддерживаемые элементом DlgTabStrip, следующие.

- **Count** – количество закладок. Установка свойства приводит к изменению количества закладок в элементе.
- **SelectedItem** – индекс выбранной (активной) закладки. Установка свойства приводит к активизации указанной закладки.
- **ItemText** – текст, который отображается на закладке. Свойство имеет один целый аргумент, который задает индекс нужной закладки.

В процессе работы элемент DlgTabStrip генерирует единственное событие.

- OnSelChanged – возникает при изменении выделенной закладки. Узнать, какая закладка активна в текущий момент, можно с помощью свойства SelectedItem.

Элемент DlgListView

Сложный список DlgListView представляет собой очень мощный элемент управления. Он похож на окна списка документов или отчетов в программах семейства *Акцент*, а также на окно списка файлов в *Windows Explorer*.

Каждый элемент списка представлен одной или несколькими текстовыми строками, необязательной картинкой и произвольным неотображаемым параметром.

Список может отображаться в одном из четырех видов: «Большие значки», «Маленькие значки», «Список» и «Таблица». Дополнительные строки каждого из элементов выводятся на экран только в режиме таблицы.

Список поддерживает поиск внутри содержимого списка.

Основные свойства, поддерживаемые списком `DlgListView`, следующие.

- **`Count`** – Свойство только для чтения. Возвращает количество элементов в списке.
- **`Columns`** – Свойство только для чтения. Возвращает количество колонок в списке. Если список не находится в режиме таблицы, то количество колонок всегда равно нулю.
- **`View`** – возвращает или устанавливает вид списка. Возможны следующие значения:
  - **`0`** Большие значки.
  - **`1`** Маленькие значки.
  - **`2`** Список.
  - **`3`** Таблица.
- **`DisplayGrid`** – если свойство равно `True`, то в режиме таблицы отображается сетка разметки.
- **`DisplayHeadings`** – если свойство равно `True`, то в режиме таблицы отображаются заголовки колонок.
- **`LabelEdit`** – если свойство равно `True`, то возможно редактирование текста элементов списка. Обратите внимание, что можно редактировать только основной текст элемента.
- **`SelectedItem`** – возвращает или устанавливает выделенный элемент списка. Если никакой элемент не выделен, возвращается значение –1. Установка свойства в –1 приводит к тому, что выделение снимается со всех элементов списка.
- **`SortEnabled`** – возвращает или устанавливает признак, разрешающий сортировку списка по произвольным колонкам. Сортировка выполняется щелчком по заголовку колонки.

Все свойства для работы с элементами списка принимают один аргумент, который задает индекс элемента списка. Индекс изменяется от 1 до `Count`. Это следующие свойства.

- **`ItemText`** – Возвращает или устанавливает текст элемента списка.
- **`ItemImage`** –возвращает или устанавливает номер картинки, отображаемой рядом с элементом списка.
- **`ItemParam`** – возвращает или устанавливает произвольный целый параметр, связанный с элементом списка.
- **`SubItemText`** – возвращает или устанавливает текст для дополнительных колонок списка. Для работы с многоколоночным списком он должен находиться в режиме таблицы (`View = 3`). Перед установкой текста для колонок их нужно добавить в список методом `InsertColumn`.

Для того чтобы отображать в списке картинки для элементов, используются два объектных свойства.

- **`ImageListBig`** – устанавливает картинку для больших значков.
- **`ImageListSmall`** – устанавливает картинки для малых значков, списка и режима таблицы.

Свойства `ImageListBig` и `ImageListSmall` устанавливаются только в режиме конструктора. Фактически, установка этих свойств означает выбор файла с картинками. Картинка должна состоять из квадратных элементов (32 x 32 пикселя для больших картинок и 16 x 16 пикселей для малых картинок). Количество изображений определяется шириной картинки. Например, изображение размером 96 x 32 пикселя представляет собой три картинки 32 x 32 пикселя. Один из цветов картинки может отображаться прозрачным. Этот цвет определяется значением свойства `MaskColor`. Обратите внимание, что цвет маски нужно выбрать до установки картинок свойствами `ImageListBig` и `ImageListSmall`.

Методы списка `DlgListView` используются для добавления/удаления элементов списка, работы с колонками, сортировки и т.д. Эти методы следующие.

- **`Clear`** – очищает список (удаляет все элементы).
- **`InsertColumn`** – Вставляет в список колонку. Метод возвращает номер вновь вставленной колонки. В качестве аргументов метода за-

дается заголовок колонки, выравнивание текста колонки и ширина колонки в пикселах.

- **InsertItem** – вставляет в список новый элемент. Метод возвращает индекс вновь созданного элемента. В качестве аргументов метода задается текст элемента и номер картинки, отображаемой рядом с элементом.
- **DeleteItem** – удаляет элемент из списка. Удаляемый элемент задается индексом. Индекс может меняться от 1 до `Count`.
- **SortItems** – сортирует содержимое списка в алфавитном порядке по заданной колонке. Обратите внимание, что сортировка выполняется даже, если свойство `SortEnabled` установлено в `False`.
- **ShowFind** – показывает стандартный диалог поиска.
- **Refresh** – просто перерисовывает список на экране.

Элемент управления `DlgListView` поддерживает события, связанные с изменением выделенного элемента, манипуляциями мышью и клавиатурой, а также с редактированием меток элементов. События, генерируемые списком `DlgListView`, следующие.

- **OnSelChanged** – возникает при изменении выделенного элемента списка. Узнать, какой элемент стал выделенным, можно с помощью свойства `SelectedItem`.
- **OnDblClick** – возникает при двойном щелчке левой кнопкой мыши. Обратите внимание, что событие возникает только при щелчке по элементу списка. Если щелкнуть по пустому месту, событие не возникает.
- **OnRClick** – возникает при щелчке правой кнопкой мыши по элементу списка.
- **OnKeyDown** – возникает при нажатии клавиши на клавиатуре, когда список имеет фокус ввода. Событие имеет два аргумента: код нажатой клавиши и набор флажков, определяющий состояние клавиш-модификаторов (Ctrl, Shift, Alt).
- **BeforeLabelEdit** – возникает перед редактированием метки элемента списка. Если свойство `LabelEdit` установлено в `False`, то редактирование меток запрещено и событие возникнуть не может.

Событие имеет аргумент `Cancel`. Если обработчик установит его в `True`, редактирование элемента будет запрещено.

- **`AfterLabelEdit`** – возникает после завершения редактирования метки элемента списка. Событие имеет аргумент `Cancel`. Если обработчик установит его в `True`, редактирование элемента будет отменено.

Элемент DlgTreeView

Элемент `DlgTreeView` представляет собой «дерево». Этот элемент управления используется для отображения многоуровневых иерархических структур данных.

Дерево содержит узлы, каждый из которых, в свою очередь, может содержать дочерние узлы, и так далее. Узел дерева представлен текстовой меткой, необязательной пиктограммой и дополнительным целым параметром, который не виден визуально и может использоваться произвольным образом.

Для ссылок на узлы дерева используется специальный тип данных `Handle` (ключ). `Handle` это длинное целое (`Long`), которое имеет значение только в контексте работы с деревом. Его конкретное значение не несет никакого смысла для программиста. Более того, никакие операции, кроме присваивания, для этого типа данных не определены.

Элемент поддерживает возможность отображения флажков (похожих на `DlgCheckBox`) в каждом узле дерева. Чтобы включить отображение флажков, установите свойство `CheckBoxes` в значение `True`.

Обратите внимание, что установка свойства во время выполнения относится только ко вновь добавляемым узлам дерева. При установке значения `False` флажки у существующих узлов остаются, но перестают срабатывать. Поэтому, смена значения свойства имеет смысл только для пустого дерева.

Для того чтобы отображать в дереве картинки для элементов, используется следующее объектное свойство.

- **`ImageList`** – устанавливает картинку для элементов дерева.

Свойство `ImageList` устанавливается только в режиме конструктора. Фактически, установка этого свойства означает выбор файла с картинка-

ми. Картинка должна состоять из квадратных элементов (16 x 16 пикселей). Количество изображений определяется шириной картинки. Например, изображение размером 48 x 16 пикселей представляет собой три картинки 16 x 16 пикселей. Один из цветов картинки может отображаться прозрачным. Этот цвет определяется значением свойства `MaskColor`.

Обратите внимание, что цвет маски нужно выбрать до установки картинок свойством `ImageList`. Установка цвета маски для уже выбранной картинки не оказывает на нее никакого влияния.

Элемент `DlgTreeView` поддерживает работу с системным списком изображений. Этот список содержит наиболее часто используемые в программе *Акцент* картинки, Для того чтобы связать дерево с таким списком, вызовите метод `SetSysImageList`.

Методы, используемые для работы с узлами дерева, следующие.

- **`InsertNode`** – вставляет узел в дерево. Метод возвращает ключ вставленного узла.

  Синтаксис

  `InsertNode``(Text[,Image][,Parent][,InsertAt])`

  Аргументы

  `Text` – текстовая метка для узла.
  `Image` – номер картинки для узла.
  `Parent` – ключ узла, в который нужно вставить узел.
  `InsertAt` – Вариант вставки. Возможны три варианта. Можно вставить узел в начало (`InsertAt = 1`), в конец (`InsertAt=2`) или вставить узел в алфавитном порядке (`InsertAt=3`).

- `DeleteNode` – удаляет заданный узел дерева.
- `ClearNodes` – полностью очищает дерево (удаляет все узлы).

Основные свойства для работы с узлами дерева следующие. Все они имеют по крайней мере один аргумент – узел для которого возвращается или устанавливается значение свойства.

- **`Root`** – возвращает корневой узел дерева.

- **SelectedNode** – возвращает ключ выделенного узла дерева. Если никакой узел не выделен, возвращается 0. При установке свойства выделяется соответствующий узел дерева.
- **ChildNode** – возвращает первый дочерний узел заданного узла дерева или 0, если таких узлов нет.
- **NextSiblingNode** – возвращает соседний узел для заданного узла дерева или 0, если заданный узел последний в списке.
- **NodeHasChildren** – возвращает `True`, если узел имеет дочерние узлы.
- **NodeParent** – свойство только для чтения. Возвращает родительский узел.
- **NodeText** – возвращает или устанавливает текстовую метку для узла.
- **NodeImage** – возвращает или устанавливает номер картинки в списке для узла дерева.
- **NodeParam** – возвращает или устанавливает произвольный параметр (типа `Long`) для узла дерева.
- **NodeChecked** – возвращает или устанавливает состояние флажка для узла. Имеет смысл только при установленном в `True` свойстве `CheckBoxes`.
- **NodeExpanded** – возвращает или устанавливает состояние дочерних элементов узла. Если свойство равно `True`, то дочерние элементы развернуты, если `False` – свернуты.

Элемент `DlgTreeView` поддерживает редактирование «на месте» для меток узлов дерева. Для того чтобы разрешить такое редактирование, установите в `True` свойство `LabelEdit`.

В процессе работы элемент `DlgTreeView` поддерживает большое количество событий. Их можно условно разделить на три группы.

События, связанные с навигацией по дереву.

- **OnNodeClick** – возникает при щелчке на узле дерева или если пользователь выделил узел каким-либо другим образом.
- **OnRClick** – возникает при щелчке правой кнопкой мыши. Обычно используется для отображения контекстных меню.

- **OnDblClick** – возникает при двойном щелчке мыши.
- **OnKeyDown** – возникает при нажатии клавиши на клавиатуре когда дерево имеет фокус ввода. Событие имеет два аргумента: код нажатой клавиши и набор флажков, определяющий состояние клавиш-модификаторов (Ctrl, Shift, Alt).

События, позволяющие контролировать процессы разворачивания / сворачивания узлов дерева.

- **CanCollapse** – возникает при попытке свернуть узел дерева. Событие имеет аргумент `Enable`. Если установить его в `False`, то узел останется развернутым.
- **CanExpand** – возникает при попытке развернуть узел дерева. Событие имеет аргумент `Enable`. Если установить его в `False`, узел останется свернутым.
- **OnCollapse** – возникает после сворачивания узла дерева.
- **OnExpand** – возникает после разворачивания узла дерева.

События, связанные с редактированием меток узлов дерева.

- **BeforeLabelEdit** – возникает перед редактированием метки узла. Если свойство `LabelEdit` установлено в `False`, то редактирование меток запрещено и событие возникнуть не может. Событие имеет аргумент `Cancel`. Если обработчик установит его в `True`, редактирование метки будет запрещено.
- **AfterLabelEdit** – возникает после завершения редактирования метки узла. Событие имеет аргумент `Cancel`. Если обработчик установит его в `True`, редактирование метки будет отменено.

Глава IX.

Электронные таблицы

В этой главе описана работа с электронными таблицами. В программах *Акцент* электронные таблицы используются как самостоятельно, так и для построения и печати различных отчетов. Приводится описание объектов электронных таблиц, их свойств, методов и событий.

Электронные таблицы *Акцент* представляют собой стандартные электронные таблицы, похожие на *Microsoft Excel*, но несколько проще. В программе они используются, в частности, для вывода стандартных отчетов. Помимо этого, электронные таблицы *Акцент* представляют собой полностью программируемые объекты. Их можно использовать для вывода произвольной информации.

Объектная модель для таблиц

Объектная модель для электронных таблиц, которая показана на рис. 5, очень проста и включает в себя всего три объекта.

Ошибка! Ошибка связи.

Рис. 5. Объектная модель для таблиц

На верхнем уровне находится *рабочая книга*, включающая один или более *рабочих листов*. Сама рабочая книга представлена объектом `ShtBook`. В ней может находиться один или более рабочих листов, каждый из которых представлен объектом `ShtSheet`.

Каждый рабочий лист представляет собой прямоугольную матрицу ячеек, как в классических электронных таблицах.

Прямоугольные области ячеек на рабочем листе описываются объектом `ShtRange` – диапазон ячеек. Диапазон может состоять, в том числе, и из одной единственной ячейки.

Электронная таблица является программируемым объектом и содержит модуль на языке *VBScript*. В этом модуле записываются все обработчики

событий. События могут генерироваться всей рабочей книгой, ее листами или кнопками панели управления.

Формулы в таблицах

В контексте модуля таблицы также выполняются выражения для формул в ячейках рабочего листа. Формулы представляют собой обычные выражения *VBScript*, в которых, на правах переменных, допустимо использование адресов ячеек. Формулы всегда начинаются со знака равенства `(=)`. Например, выражение `=A1+B1` означает суммирование значений ячеек `A1` и `B1`.

Помимо ссылок на ячейки таблицы, в формулах можно использовать встроенные функции. Их список приведен ниже.

Функция	Синтаксис	Описание
SUM	`SUM(A1:B2)`	Сумма ячеек диапазона. A1 и B2 представляют собой адреса верхней левой и нижней правой ячеек диапазона.

Создание таблицы

В программах семейства *Акцент* создание таблицы возможно двумя способами. Во-первых, таблицу можно создать вызовом метода `CreateBook` объекта `Application`.

Во-вторых, все объекты отчетов, имеющие метод `MakeSheet`, позволяют выдать результат построения отчета в таблицу и получить ссылку на нее.

Такой подход позволяет программисту дополнительно обработать отчет перед тем, как показать его пользователю.

Метод `CreateBook` имеет следующий синтаксис.

Синтаксис

```
Set sht = app.CreateBook([Book],[Visible])
```

Аргументы

`Book` – Необязательный аргумент типа `Variant`. Если это целое число, то оно рассматривается как идентификатор таблицы, подключенной к проекту. Если это строка, то она рассматривается как имя файла таблицы `*.ASH`. Если путь к файлу не указан, учитывается информация о расположении файлов из

*.ARD. Если аргумент опущен, то создается новая чистая электронная таблица.

Visible – Необязательно. Если значение аргумента равно False, то таблица создается невидимой. Для ее показа на экране нужно установить в True ее свойство Visible.

Метод MakeSheet для объектов отчетов также имеет необязательный аргумент – Visible . Если его значение равно False, то таблица создается невидимой. Для ее показа на экране нужно установить в True ее свойство Visible.

Не забывайте о том, что метод возвращает ссылку на объект, поэтому использование ключевого слова Set является обязательным.

Объект ShtBook

Объект ShtBook определяет верхний уровень иерархии в объектной модели электронных таблиц. Внутри объекта ShtBook находятся все листы таблицы (ShtSheet). Кроме того, в этом объекте есть карта Map и пользовательская панель инструментов Toolbar. Ссылки на них возвращаются соответствующими свойствами.

Основные свойства объекта ShtBook следующие.

- **ActiveSheet** – возвращает ссылку на текущий активный рабочий лист.
- **Caption** – текстовая строка, определяющая заголовок окна рабочей книги.
- **Map** – возвращает ссылку на карту Map.
- **PeriodText** – текстовая строка, которая будет выводиться в правой части строки состояния окна. Обычно там выводится наименование текущего отчетного периода.
- **ReadOnly** – если это свойство имеет значение True, то редактирование таблиц невозможно.
- **Sheet** – обеспечивает доступ к листам таблицы. Работает как по индексам, так и по именам.
- **StatusBarVisible** – если равно True, то в нижней части окна выводится строка состояния.

- **ToolbarVisible** – если равно `True`, то выводится стандартная панель инструментов, на которой расположены кнопки наиболее часто используемых команд. Обратите внимание, что у таблицы, как и у формы, имеется пользовательская панель инструментов, которая автоматически выводится на экран, если на ней есть хотя бы одна кнопка.
- **Toolbar** – обеспечивает доступ к пользовательской панели инструментов (объект `Toolbar`).
- **Topmost** – если это свойство равно `True`, то окно таблицы всегда располагается поверх остальных.
- **Visible** – если свойство равно `True`, то таблица видима, если `False` – то скрыта.
- **Modified** – свойство возвращает `True`, если таблица изменялась и требуется ее сохранение на диске.
- **PreTranslate** – свойство позволяет управлять режимом «перехвата» формул. Если оно установлено в True, то перед вычислением формулы возникает событие `OnTranslate`. Обработчик события может взять вычисления на себя и вернуть готовый результат.

Для выполнения действий со всей рабочей книгой используются следующие методы.

- **AddSheet** – добавляет к книге новый рабочий лист.
- **RemoveSheet** – удаляет рабочий лист из таблицы.
- **Close** – закрывает окно таблицы.
- **Cut, Copy, Paste** – обеспечивают работу с буфером обмена *Windows*. Работают с активным рабочим листом таблицы и с выделенным диапазоном ячеек. Для управления выделением ячеек можно воспользоваться методом `Select` объекта `ShtRange`.
- **Maximize** – распахивает окно таблицы во весь экран.
- **Minimize** – сворачивает окно таблицы в значок.
- **GetScript,SetScript** – позволяют получить или записать текст модуля таблицы.
- **FireUserEvent** – обеспечивает генерацию события `OnUser`.

- **Move** – перемещает окно таблицы по экрану и/или изменяет размеры окна.
- **Recalc** – пересчитывает активный рабочий лист.
- **Save,SaveAs** – сохраняют изменения в файле, возможно с новым именем.
- **Preview** – показывает окно предварительного просмотра перед печатью.
- **Print** – печатает активный лист таблицы. Метод имеет один необязательный аргумент `ShowDialog`. Если его значение равно `True`, то выводится стандартный диалог печати.
- **ShowPopup** – позволяет показать на экране всплывающее меню. При выборе команды из такого меню генерируется событие `OnPopup`.
- **EnableButtons** – позволяет разрешить или запретить работу некоторых кнопок стандартной панели инструментов.
- **ExecBarTag** – рассматривает значение свойства `Tag` кнопки панели инструментов как инструкцию *VBScript* и выполняет ее. Предназначено для упрощения написания обработчика события `OnBarClick`.

В процессе своей работы объект `ShtBook` может получать следующие события.

- **BeforePrint,AfterPrint** – возникают перед/после печати рабочего листа.
- **BeforeRecalc,AfterRecalc** – возникают перед/после перерасчета таблицы. Позволяют контролировать процесс пересчета таблицы.
- **OnActivate** – возникает при активизации какого-либо рабочего листа. Активный рабочий лист передается в процедуру обработки события в качестве аргумента.
- **OnBarClick** – возникает при нажатии кнопки на пользовательской панели инструментов.
- **OnBarDropDown** – возникает при нажатии «выпадающей» кнопки на пользовательской панели инструментов.

- **OnPopup** – возникает при выборе команды из всплывающего меню, которое может быть вызвано методом ShowPopup или с помощью объекта PopupMenu.
- **OnPeriodClick** – возникает при щелчке мышью на панели периода в строке состояния.
- **OnRequery,OnWhatThis** – возникают при нажатии кнопок «Обновить» и «Что это значит?» на стандартной панели инструментов.
- **OnLoad** – возникает после загрузки таблицы, но до появления ее на экране.
- **OnUser** – возникает при вызове метода FireUserEvent.
- **OnTranslate** – возникает перед вычислением формулы. Для того, чтобы вырабатывалось это событие, нужно, чтобы значение свойства PreTranslate было установлено в True.

Объект ShtSheet

Объект ShtSheet представляет собой рабочий лист в рабочей книге. Внутри объекта ShtSheet находятся все ячейки таблицы. Кроме того, в этом объекте есть объект PageSetup, предназначенный для управления параметрами печатной страницы. Ссылка на него возвращается соответствующим свойством PageSetup.

Доступ к ячейкам таблицы основывается на понятии *диапазона*. Диапазон – это прямоугольный участок ячеек. Диапазон может состоять, в том числе, и из одной ячейки.

Применение диапазонов имеет одну особенность. Когда Вы читаете значение какого-либо свойства диапазона, система возвращает значение, связанное с левой верхней ячейкой диапазона. В процессе записи, однако, нужное свойство присваивается всем ячейкам диапазона.

Основные свойства объекта ShtSheet следующие.

- **Caption** – заголовок листа. Заголовок отображается на ярлычке, который выводится в нижней части окна рабочей книги.
- **Rows,Columns** – количество строк и столбцов в таблице.
- **DisplayGrid,DisplayHeadinghs** – отображать или нет сетку и заголовки строк и столбцов.

- **FixRows,FixColumns** – количество строк и колонок, которые «зафиксированы» от прокрутки.
- **HScroll,VScroll** – положение полос прокрутки. Фактически это адрес первой видимой ячейки таблицы. Свойства можно устанавливать. Это приводит к программной прокрутке таблицы.
- **Name** – наименование листа. Служит для ссылок на этот лист в программе.
- **Selection** – возвращает диапазон выделенных ячеек таблицы. Если ничего не выделено, то возвращается одна ячейка таблицы – та, на которой стоит курсор.
- **Cell** – возвращает ссылку на одну ячейку. Хотя это и одна ячейка, она все равно представляет собой диапазон (объект `ShtRange`).
- **EntireRow,EntireColumn** – возвращают ссылку на диапазон, равный целой строке или колонке.
- **Range** – возвращает ссылку на заданный диапазон. Имеет четыре аргумента, задающие соответственно левую, верхнюю, правую и нижнюю ячейки диапазона.
- **Index** – текущий номер в списке рабочих листов книги.
- **PageSetup** – возвращает ссылку на объект `PageSetup`, связанный с управлением параметрами печатной страницы.
- **Visible** – если свойство равно `True`, то лист видим, если `False` – то скрыт. Обратите внимание, что хотя бы один лист в книге должен быть виден всегда.

Объект `ShtSheet` поддерживает следующий список методов.

- **Activate** – активизирует данный лист. Родительская рабочая книга получит событие `OnActivate`.
- **AutoFit** – подгоняет ширину и высоту ячеек всего листа по их содержимому. Объект диапазона `ShtRange` также имеет свойство `AutoFit`, которое работает только в пределах диапазона.
- **CellAt** – возвращает адрес ячейки, которая находится на месте с заданными координатами. Координаты задаются в пикселях относительно левого верхнего угла рабочей области окна. Чаще всего метод используется в обработчике события `OnContextMenu`.

- **DeleteRow**, **DeleteColumn** – удаляет строку или колонку.
- **InsertRow**, **InsertColumn** – вставляет новую строку или колонку.
- **PageBreak** – управляет переводами страниц при печати.

В процессе своей работы объект ShtSheet может генерировать следующие события.

Напомним, что хотя обработчики этих событий и находятся в модуле рабочей книги, префиксом для них будет имя рабочего листа. Например, если лист имеет имя Sheet1, то обработчиком события OnClick для этого листа будет процедура Sheet1_OnClick.

- **OnActivate** – возникает при активизации рабочего листа.
- **OnCellChanged** – возникает при любых изменениях содержимого ячеек. В процедуру обработки события передаются координаты измененной ячейки.
- **OnClick** – возникает при щелчке левой кнопкой мыши на рабочем листе.
- **OnContextMenu** – возникает при нажатии правой кнопки мыши на листе. В качестве аргументов обработчик события получает координаты курсора мыши.

Объект ShtRange

Объект ShtRange представляет собой прямоугольный участок ячеек на рабочем листе в рабочей книге.

Применение диапазонов имеет одну особенность. Когда Вы читаете значение какого-либо свойства диапазона, программа возвращает значение, связанное с левой верхней ячейкой диапазона. В процессе записи, однако, нужное свойство присваивается всем ячейкам диапазона.

Основные свойства объекта ShtRange следующие.

- **Alignment,VAlignment** – выравнивание текста по горизонтали и по вертикали.
- **BackColor,ForeColor** – цвет фона и цвет текста.
- **ColumnWidth,RowHeight** – ширина колонок и высота строк в пикселях.

- **DataType –** тип данных, которые хранятся в ячейках диапазона. Поддерживаемые значения следующие:

vbLong	целое число
vbString	текстовая строка
vbDouble	число с плавающей точкой
vbCurrency	денежное значение
vbDate	дата/время

- **Font –** шрифт. Представляет собой стандартный объект `Font`.
- **Height,Width –** высота и ширина диапазона (в колонках и строках).
- **MergeCells –** если это свойство равно `True`, то все ячейки диапазона объединяются в одну большую ячейку.
- **Multiline –** если это свойство равно `True`, то разрешен перенос текста по словам.
- **Left, Top –** левая и верхняя координаты диапазона (в колонках и строках).
- **Value** – значение, которое хранится в ячейке. Тип значения совпадает с типом данных, которые хранятся в ячейке (определяется значением свойства `DataType`).

Основные методы, поддерживаемые объектом `ShtRange`.

- **AutoFit –** подгоняет ширину и высоту ячеек по их содержимому. При этом учитываются объединенные ячейки. Метод имеет необязательные флажки, которые позволяют подгонять либо только ширину, либо только высоту ячеек.
- **Equalize–** выравнивает ширину и/или высоту ячеек по большей из них.
- **Select –** выделяет диапазон.
- **SetBorder –** устанавливает рамки для диапазона.
- **SetDateFormat,SetNumFormat,SetPrecision –** устанавливают различные характеристики форматирования данных для ячеек, такие как формат даты, числа и точность для отображения чисел и сумм.

- **`Stripe`** раскрашивает диапазон «в полосочку». Метод имеет два необязательных аргумента, которые задают цвет фона для четных и нечетных строк. Если аргументы опущены, то берутся стандартные цвета (белый и светло-голубой). Конкретные значения этих цветов можно получить с помощью свойства `StdBackColor` объекта `ShtBook`.

Глава X.

Доступ к данным (DAO)

В этой главе описаны основные объекты *DAO*, используемые для непосредственного доступа к базе данных в настольных редакциях программ *Акцент*.

Объекты доступа к данным *Data Access Objects (DAO)* корпорации *Microsoft®* используются для доступа к базам данных практически во всех офисных приложениях. Кроме поддержки фирменной технологии *Microsoft Jet*, объекты *DAO* поддерживают доступ практически к любым базам данных, включая *DBF*, *Paradox*, *Visual FoxPro* и т.д.

Настольные версии программ семейства *Акцент* используют *DAO* для доступа к базе данных приложения. В версиях клиент-сервер используется другая технология доступа к данным – *ActiveX Data Objects* (*ADO*). Технология *ADO* описана в следующей главе.

Приведенное описание ни в коей мере не претендует на полноту. Для получения более подробных сведений обратитесь к документации корпорации *Microsoft®*.

Объектная модель DAO

На рисунке 6 приведена упрощенная объектная модель *DAO*. Как видно из рисунка, модель является «иерархической». Это означает, что все объекты вложены друг в друга. Никакой объект невозможно получить непосредственно, минуя всю последовательность вложенных объектов.

Практически все объекты *DAO* являются членами соответствующих коллекций. Например, открытые базы данных, представленные объектом `Database`, входят в коллекцию `Databases`. Как и во многих других случаях, одиночный объект обозначается существительным в единственном числе, а коллекция таких объектов – во множественном.

Ошибка! Ошибка связи.

Рис. 6. Упрощенная объектная модель DAO

Исключение составляет объект `DBEngine`, который существует в единственном экземпляре. Он представляет собой непосредственно среду *DAO* – «машину баз данных».

Далее будет приведено краткое описание каждого элемента этой объектной модели.

- **`DBEngine`** – представляет собой машину баз данных *DAO*. Содержит в себе все остальные объекты. Используется для обращения к ядру базы данных, как к единому целому.
- **`Workspace`** и **`Workspaces`** – рабочая область (коллекция). Используются при построении защиты на уровне пользователя. Кроме того, все базы данных, открытые в одной рабочей области, используют одно и то же пространство транзакций.
- **`Database`** и **`Databases`** – представляет собой базу данных. Коллекция `Databases` содержит все открытые в текущий момент базы данных.
- **`TableDef`** и **`TableDefs`** – коллекция `TableDefs` содержит все существующие в базе данных таблицы. Таблицы представлены объектом `TableDef`.
- **`QueryDef`** и **`QueryDefs`** – коллекция `QueryDefs` содержит все сохраненные в базе данных запросы. Каждый запрос представлен объектом `QueryDef`. Следует отметить, что можно создавать временные запросы, которые не нужно добавлять в базу данных.
- **`Recordset`** и **`Recordsets`** – объект `Recordset` представляет собой набор записей, полученных в результате выполнения запроса к базе данных. Все открытые наборы попадают в коллекцию `Recordsets`. Когда набор закрывается, он автоматически удаляется из коллекции.
- **`Field`** и **`Fields`** – коллекция `Fields`, которая может быть связана с объектами `TableDef`, `QueryDef` и `Recordset` содержит поля (столбцы) таблицы, запроса или набора записей. Каждое поле представляет собой объект `Field`.

- **`Parameter`** и **`Parameters`** – коллекция связана только с запросом и содержит параметры, которые необходимы для параметризованных запросов. Каждый параметр представлен объектом `Parameter`.

Примечание. Обратите внимание, что во всех коллекциях *DAO* элементы нумеруются с нуля, а не с единицы, как во всех остальных случаях.

Доступ к DAO из Акцент

Поскольку настольные версии программ семейства *Акцент* используют DAO в качестве машины баз данных, постольку имеется возможность предоставить программисту доступ к внутренним объектам DAO, уже используемым программой.

Для доступа к текущей открытой базе данных служит свойство `DaoDatabase` объекта `Workarea`. Оно возвращает ссылку на объект `Database`.

Если нужно получить доступ к ядру, можно воспользоваться свойством `DaoEngine` объекта `Workarea`, которое возвращает ссылку на объект `DBEngine`.

Следует отметить, что можно конечно создать объект `DBEngine` вызовом стандартной функции `CreateObject`, но связать его с текущей открытой базой данных будет невозможно.

Константы DAO и файл DAO.INC

Многие свойства объектов *DAO* задаются специальными константами. Эти константы обычно начинаются с префикса `db`. Эти константы НЕ включены в программы *Акцент*. Для того чтобы использовать эти константы, их нужно определить инструкцией `Const`.

В комплект поставки *Акцент* входит включаемый файл `DAO.INC`, в котором содержатся наиболее часто используемые константы *DAO*. Включите этот файл в свой модуль с помощью директивы `#include`.

Объекты DAO

Объект DBEngine

Объект `DBEngine` представляет собой ядро машины баз данных *DAO*. Он нужен тогда, когда нужно обратиться к системе управления базами данных как к единому целому.

Объект `DBEngine` поддерживает следующие основные свойства.

- **`Version`** – номер версии *DAO*. У объекта `Database` тоже есть свойство `Version`, но оно возвращает версию конкретной базы данных.
- **`IniPath`** – путь к файлу системной базы данных `SYSTEM.MDW`. Эта база данных используется для организации защиты на уровне пользователей.
- **`LoginTimeout`** – интервал в секундах, в течение которого система будет ожидать соединения с базой данных. Имеет смысл только для *ODBC*-подключений.
- **`DefaultUser`** и **`DefaultPassword`** – имя пользователя и пароль по умолчанию, который используется при инициализации ядра *Jet*. Следует отметить, что пароль устанавливается на всю систему управления базами данных (фактически на доступ к *DAO*), а не на конкретную базу данных.
- **`Workspaces`** – возвращает коллекцию открытых рабочих областей `Workspaces`.
- **`Errors`** – возвращает коллекцию ошибок (`Errors`), которые произошли в системе.

Объект `DBEngine` имеет следующие методы.

- **`CreateWorkspace`** – создает новый объект Workspace.
- **`CompactDatabase`** – выполняет сжатие базы данных, копируя ее в новый файл. Все лишние данные удаляются, что повышает быстродействие.
- **`RepairDatabase`** – пытается восстановить поврежденную базу данных. Файл базы данных может быть поврежден в результате сбоев оборудования.

Объект Workspace и коллекция Workspaces

Объекты `Workspace` используются в основном для управления доступом к базам данных на уровне пользователей. Коллекция `Workspaces` не бывает пустой. После инициализации ядра `DBEngine` она содержит одну рабочую область с индексом 0 и именем `"#DefaultWorkspace#"`. В программах семейства *Акцент* всегда используется рабочая область по умолчанию. В этой книге мы ограничимся лишь упоминанием об объектах `Workspace`. Для получения дополнительной информации обратитесь к документации *Microsoft*.

Объект Error и коллекция Errors

Коллекция `Errors` содержит ошибки (представленные объектом `Error`), которые произошли при работе с базой данных. Обратите внимание, что если обработка ошибок в программе выключена, то коллекция всегда будет пустой, поскольку при возникновении ошибки, программа просто прервется. Коллекция имеет стандартные свойства `Count` и `Item`, используемые для доступа к ее элементам.

Объект `Error` содержит описание ошибки. Он имеет следующие свойства (все они только для чтения).

- **`Description`** – текстовое описание ошибки.
- **`HelpFile`** – полный путь к файлу справки, который содержит описание данной ошибки.
- **`HelpContext`** – номер раздела справочной системы, содержащий информацию об ошибке.
- **`Number`** – внутренний код ошибки.
- **`Source`** – имя объекта, ставшего источником ошибки.

Объект Database и коллекция Databases

Коллекция `Databases` содержит все открытые на текущий момент базы данных. Каждая такая база данных представлена объектом `Database`. Он представляет собой открытую базу данных. Ссылку именно на этот объект возвращает свойство `DaoDatabase` объекта `Workarea`.

Коллекция `Databases`, как и все коллекции, имеет свойство `Count`, возвращающее количество элементов, и свойство `Item`, позволяющее получить ссылку на элемент коллекции по номеру или по имени.

Можно открыть вторую базу данных в текущей рабочей области, воспользовавшись методом `OpenDatabase` объекта `Workspace`.

Внутри объекта `Database` содержатся следующие коллекции

- **`TableDefs`** – все таблицы базы данных
- **`QueryDefs`** – все запросы, которые записаны в базу данных
- **`Recordsets`** – все открытые в текущий момент наборы записей.

Ссылки на эти коллекции возвращаются соответствующими одноименными свойствами.

Объект **`Database`** обладает следующими основными свойствами.

- **`Name`** – имя базы данных. Доступно только для чтения.
- **`Version`** – версия базы данных. Фактически это версия *DAO*, под управлением которой она создана. Значение этого свойства вполне может отличаться от значения свойства `Version` объекта `DBEngine`.
- **`QueryTimeout`** – время, в течение которого система будет ожидать завершения выполнения запросов.
- **`RecordsAffected`** – количество записей, которые были «затронуты» при выполнении последнего запроса. Доступно только для чтения.
- **`Updatable`** – если базу данных можно изменять, значение свойства равно `True`.
- **`Transactions`** – равно `True`, если база данных поддерживает транзакции.

Основные действия, которые можно выполнить с базой данных, описываются следующими методами.

- **`OpenRecordset`** – открывает набор записей, выполняя запрос к базе данных. Возвращает объект `Recordset`. Набор записей может быть получен различными способами из объектов `Database`, `QueryDef` и `TableDef`. Работа с наборами записей будет описана ниже.
- **`Execute`** – выполняет запрос к базе данных, который не возвращает результатов. Обычно это запрос на добавление, изменение или

удаление записей. Запросы с параметрами этим методом не поддерживаются. Чтобы выполнить параметризованный запрос, воспользуйтесь объектом QueryDef.

Пример

```
Dim db
Dim sql
Set db = Workarea.DaoDatabase
sql = "update TABLE set FIELD=1"
db.Execute sql
```

- **CreateQueryDef** – создает временный объект запроса QueryDef и возвращает ссылку на него. Вновь созданный объект НЕ добавляется в коллекцию QueryDefs. Чтобы сохранить его в базе данных, нужно добавить его в коллекцию методом Append.

Объект Recordset

Объект Recordset является основным объектом, обеспечивающим доступ к данным. Он позволяет получить информацию из базы данных в виде таблицы с последовательным доступом. Такая таблица состоит из строк (записей) и столбцов (полей). Объект Recordset может использоваться для навигации по записям набора, а также для добавления, изменения и удаления записей.

Виды объектов Recordset

Базы данных *Jet* поддерживают несколько видов объектов Recordset.

- Table – табличный. Используется для доступа к данным из одной таблицы. Возможно перемещение по записям вперед и назад, добавление, изменение и удаление записей.

- Dynaset – динамический. Может представлять данные из нескольких связанных таблиц. Поддерживает перемещение по записям в обоих направлениях и ограниченные возможности по добавлению, изменению и удалению записей.

- Snapshot – статический. Используется только для чтения данных. Дополнительно может иметь атрибут *«forward only»* (только вперед). Это самый быстрый и наименее ресурсоемкий вид набора.

Объект Field и коллекция Fields

Все столбцы (поля) набора записей собраны в специальную коллекцию `Fields`, которая содержится в объекте `Recordset`. Ссылка на нее возвращается одноименным свойством `Fields`. В коллекции находятся объекты `Field`, каждый из которых представляет собой одно поле из набора. Доступ к полям возможен как по порядковым номерам (не забывайте, что в DAO коллекции нумеруются с нуля!), так и по именам.

Как и все коллекции, `Fields` содержит свойство `Count`, возвращающее количество элементов, и свойство `Item`, возвращающее элемент коллекции по его имени или индексу.

Основные свойства объекта `Field` следующие.

- **`Name`** – имя поля.
- **`Type`** – тип данных, которые хранятся в поле. Конкретные типы определяются специальными константами. Их можно посмотреть в файле `DAO.INC` из комплекта поставки.
- **`Value`** – значение поля (тип `Variant`). Свойство является свойством «по умолчанию» и его можно опускать.
- **`Size`** – длина поля, как она определена в базе данных.
- **`FieldSize`** – реальная длина данных, которые хранятся в поле.

Пример

```
Dim f     ' объектная переменная (для поля)
Dim name  ' имя поля
Dim value ' значение поля
' проход по всем записям
While Not rs.EOF
   For Each f In rs.Fields
      name = f.Name
      value = f.Value
      ' что-то делаем
   Next
   rs.MoveNext
Wend
```

Доступ к Recordset

Получить ссылку на объект Recordset можно несколькими способами. Во первых, ее можно получить, вызвав метод OpenRecordset объекта Database.

Во вторых, можно получить объект, представляющий записи одной таблицы вызвав метод OpenRecordset объекта TableDef. Если нужно выполнить запрос с параметрами, то для этого опять используется метод OpenRecordset, но уже объекта QueryDef.

В зависимости от объекта, к которому применяется вызов, синтаксис вызова метода OpenRecordset следующий.

Синтаксис для Database

```
db.OpenRecordset(Src,[Type],[Options],[Locks])
```

Синтаксис для QueryDef

```
db.OpenRecordset([Type],[Options],[Locks])
```

Аргументы

Src – источник данных. Может быть именем таблицы, именем сохраненного запроса или инструкцией *SELECT*.

Type – тип набора (подробности см. в справочной системе).

Options – характеристики набора записей (подробности см. в справочной системе).

Locks – способы блокировки записей в многопользовательском режиме (подробности см. в справочной системе).

Навигация по объекту Recordset

После того как набор записей открыт, по нему можно перемещаться вперед и, возможно, назад (в зависимости от типа набора). Навигация заключается в том, что воображаемый указатель (называемый также *текущей записью*) перемещается по набору данных.

Вначале рассмотрим два свойства, определяющие позицию текущей записи в наборе.

- EOF –становится равным True, если текущая запись находится непосредственно за последней записью набора.
- BOF – значение True говорит о том, что текущая запись находится перед первой записью набора данных.

Для перемещения текущей записи служат следующие методы

- `MoveNext` – на одну запись вперед.
- `MovePrevious` – на одну запись назад.
- `MoveFirst` – на первую запись.
- `MoveLast` – на последнюю запись.

Очевидно, что для наборов типа *«forward only»*, попытка выполнить методы `MovePrevious` и `MoveFirst` приведет к ошибке.

При работе с этими методами всегда нужно анализировать положение текущей записи (свойства `EOF` и `BOF`). Попытка перемещения за пределы набора записей приведет к ошибке выполнения программы.

Примеры

```
' проход по всем записям набора
While Not rs.EOF
   ' что-то делаем
   rs.MoveNext
Wend

' проверка набора на пустоту
If rs.EOF And rs.BOF Then
   MsgBox "Нет записей"
End If
```

Добавление, редактирование и удаление записей

Объекты `Recordset` табличного и динамического типа поддерживают возможность добавления, изменения и удаления записей.

Если текущий набор поддерживает изменения в данных, то его свойство `Updatable` имеет значение `True`.

Методы, которые используются для изменения данных, следующие.

- `AddNew` – добавляет новую пустую запись в конец набора.
- `Edit` – делает возможным редактирование текущей записи.
- `Update` – записывает произведенные изменения в базу данных.
- `CancelUpdate` – отменяет запись изменений.

- `Delete` – удаляет текущую запись из набора. Не требует подтверждения.

После того как `Recordset` установлен в режим изменения данных (после вызова `AddNew` или `Edit`), можно изменять данные, устанавливая значения свойства `Value` объекта `Field`. Объект `Field` представляет собой поле (столбец) данных. Все поля содержатся в коллекции `Fields`, которая возвращается свойством `Fields`. Доступ к полям возможен как по именам, так и по порядковым номерам.

Примеры

```
' добавление записи
Dim sql
sql = "SELECT F1,F2 FROM T1"
Set rs = db.OpenRecordset(sql)
With rs
   .AddNew
   .Fields(0).Value = "value1"
   .Fields(1).Value = "value2"
   .Update
End With

' изменение записи
Dim sql
sql = "SELECT F1,F2 FROM TEST WHERE ID=1"
Set rs = db.OpenRecordset(sql)
With rs
   .Edit
   .Fields("F1").Value = "field1"
   .Fields("F2").Value = "field2"
   .Update
End With
```

Объект QueryDef

Объект `QueryDef` представляет собой запрос на языке *SQL* (возможно с параметрами). Запрос можно сохранить в базе данных с определенным именем и выполнять его по мере необходимости.

Доступ к объектам QueryDef

Для того чтобы получить ссылку на сохраненный в базе данных запрос, нужно обратиться к коллекции QueryDefs. Эта коллекция возвращается одноименным свойством QueryDefs объекта Database.

Как и все коллекции, QueryDefs имеет свойство Count, возвращающее количество элементов и свойство Item, позволяющее получить ссылку на элемент как по индексу или по имени.

Пример

```
Dim qd
Set qd = db.QueryDefs("my_query1")
```

Возможно создание временных объектов QueryDef, которые не имеют имени и уничтожаются после использования. Заметим, что это единственный способ выполнения параметризованных запросов, которые не сохранены в базе данных.

Для того чтобы создать временный объект QueryDef, нужно воспользоваться методом CreateQueryDef объекта Database. Методу передается два аргумента: имя (для временного объекта может быть пустым) и *SQL*-выражение текста запроса. Заметим, что текст запроса можно установить и позже с помощью свойства SQL.

Пример

```
Dim qd, sql
sql = "PARAMETERS [ID] LONG" & _
      "SELECT * FROM TEST WHERE ID=[ID]"
Set qd = db.CreateQueryDef("",sql)
```

Если объект QueryDef предполагается сохранить на будущее, то его нужно добавить к коллекции QueryDefs с помощью метода Append. Разумеется, что в этом случае он должен иметь непустое имя.

Пример

```
Dim qd
Dim sql
sql = "PARAMETERS [ID] LONG" & _
      "SELECT * FROM TEST WHERE ID=[ID]"
Set qd = db.CreateQueryDef("my_query",sql)
Call db.QueryDefs.Append(qd)
```

Объект Parameter и коллекция Parameters

Для того чтобы можно было использовать параметры в запросах, используется коллекция Parameters и содержащиеся в ней объекты Parameter.

Ссылка на коллекцию возвращается одноименным свойством Parameters объекта QueryDef.

Как и все коллекции, Parameters имеет свойство Count, возвращающее количество элементов и свойство Item, позволяющее получить ссылку на элемент (Parameter) как по индексу в коллекции, так и по имени параметра, заданному в строке запроса.

Основные свойства объекта Parameter следующие.

- **Name** – имя параметра.
- **Type** – тип данных параметра.
- **Value** – значение параметра (тип Variant). Свойство является свойством «по умолчанию» и его можно опускать.
- **Size** – длина поля, как она определена в базе данных.

Пример

```
Dim qd
Dim rs
Dim sql
sql = "PARAMETERS [ID] LONG" & _
      "SELECT * FROM TEST WHERE ID=[ID]"
Set qd = db.CreateQueryDef("",sql)
qd.Parameters(0).Value = 1
' или qd.Parameters("[ID]").Value = 1
Set rs = qd.OpenRecordset
While Not rs.EOF
   ' что-то делаем
   rs.MoveNext
Wend
```

Свойства и методы QueryDef

Основные свойства объекта QueryDef следующие.

- **Name** – имя запроса.

- **SQL** – SQL-выражение текста запроса.
- **RecordsAffected** – число записей, которые «затронул» запрос в процессе последнего выполнения.
- **ReturnsRecords** – указывает, будет ли запрос возвращать данные.

Основные методы объекта `QueryDef`.

- **Execute** – выполняет запрос на вставку, изменение или удаление данных. Не возвращает никакого результата.
- **OpenRecordset** – выполняет запрос на выборку данных и возвращает объект `Recordset`.

Глава XI.

Доступ к данным (ADO)

В этой главе описаны основные объекты *ADO*, используемые для непосредственного доступа к базе данных в редакциях «клиент-сервер», программ *Акцент*.

Технология *ActiveX Data Objects (ADO)* одна из наиболее мощных и современных технологий для доступа к информации. Фактически *ADO* является высокоуровневой объектно-ориентированной «оберткой» для универсальной технологии доступа к данным от *Microsoft*, которая носит имя *OLE DB*. Технология *OLE DB* обеспечивает практически одинаковый доступ к *любым* (а не только реляционным) данным. Для этого нужно лишь иметь соответствующий драйвер (называемый в *OLE DB* «поставщиком данных» – *provider*).

В программах *Акцент* объекты *ADO* используются для предоставления программисту прямого (на уровне языка *SQL*) доступа к базе данных в вариантах поставки программы «клиент-сервер» (*Акцент – MSDE* и *Акцент – SQL*).

Следует отметить, что приведенное описание ни в коем случае не претендует на полноту. Здесь будут описаны только наиболее часто используемые объекты, их свойства и методы. Для получения более полной информации обращайтесь к справочной системе по ADO, поставляемой корпорацией *Microsoft*.

Сами программы *Акцент* не используют *ADO*. Они работают с использованием более низкоуровневой (и более быстрой) технологии *OLE DB*.

Следует отметить, что работа с объектами *ADO* существенно отличается от *DAO*. Все объекты ADO независимы друг от друга и могут создаваться сами по себе. Для связи объектов между собой используются специальные свойства объектов. Например, для того, чтобы получить ссылку на объект `QueryDef`, представляющий запрос к базе данных в *DAO*, нужно обратиться к коллекции `QueryDefs`, которая, в свою очередь, возвращается специальным свойством объекта `Database`. В *ADO* нужно просто соз-

дать объект Command (стандартной функцией CreateObject), а затем присвоить его свойству ActiveConnection ссылку на существующий объект Connection.

Фактически, если объекты *DAO* организованы иерархически и связи между ними жестко установлены, то в *ADO* наоборот, объектная модель является плоской, а связи устанавливаются по мере необходимости.

Объектная модель ADO

На рисунке 7 приведена упрощенная объектная модель *ADO*. Как видно из рисунка, модель достаточно проста и содержит всего несколько объектов.

Ошибка! Ошибка связи.

Рис. 7. Объектная модель ADO (упрощенная)

Далее будет приведено краткое описание каждого элемента этой объектной модели.

- **Connection** – представляет собой соединение между источником данных и приложением. Для того чтобы все остальные объекты могли получить доступ к данным, они должны быть предварительно связаны с объектом Connection.
- **Command** – описывает команду (запрос), которая будет передана для обработки поставщику данных. Объекты Command используются либо для получения наборов записей (объектов Recordset), либо для выполнения запросов на вставку, удаление или обновление записей.
- **Parameter** – описывает содержащийся в команде параметр, предназначенный для передачи информации поставщику данных. Параметры группируются в коллекцию Parameters. Если запрос, содержит параметры, то перед выполнением запроса они должны быть созданы и добавлены в коллекцию параметров.
- **Recordset**—представляет собой набор записей, в котором непосредственно содержатся данные. По сути, с помощью этого объекта выполняется большинство действий над данными.
- **Field** – описывает отдельный столбец (поле) в наборе записей. Позволяет получать и изменять данные, а также получать информацию о размере и свойствах полей объекта Recordset. Поля груп-

пируются в коллекцию Fields. Объект Field нельзя создать независимо, он всегда принадлежит набору данных Recordset.

- **Error** – описывает ошибку, которая могла произойти при выполнении операции. Объект Connection содержит коллекцию Errors, в которой хранятся все возникшие в процессе работы ошибки.

Доступ к ADO из Акцент

Поскольку *Акцент* работает с базой данных на уровне *OLE DB*, то невозможно предоставить непосредственный доступ к *ADO*. Можно было бы просто создавать объект ADODB.Connection стандартной функцией CreateObject, но в этом случае нельзя будет получить доступ к текущей базе данных. Если просто установить такое же соединение, как и для запуска основной программы (через файл UDL), то возникает проблема паролей. Либо их нужно хранить в UDL файле (что крайне не рекомендуется), либо пользователь вынужден будет вводить пароль при каждом подключении к базе данных, что, конечно, неприемлемо.

Предоставление программного доступа к строке подключения *OLE DB*, содержащей имя пользователя и пароль также неприемлемо из соображений безопасности.

Для решения описанных проблем в программах *Акцент* используется следующий подход. Объект Workarea содержит специальное свойство AdoConnection, которое автоматически создает объект ADODB.Connection, устанавливает соединение с текущей открытой базой данных с использованием текущего имени пользователя и пароля и возвращает ссылку на созданный объект.

Например, чтобы выполнить запрос к базе данных, нужно проделать следующие действия:

```
Dim cnn  ' активное соединение
Dim cmd  ' объект – команда
Dim rst  ' набор для получения записей

' получим Connection
Set cnn = Workarea.AdoConnection

' созадим команду
Set cmd = CreateObject("ADODB.Command")
```

```
' свяжем команду с Connection
Set cmd.ActiveConnection = cnn

' установим текст команды
cmd.CommandText = "select * from TABLE"

' выполним запрос, получив Recordset
Set rst = cmd.Execute
```

Константы ADO и файл ADO.INC

Многие свойства объектов *ADO* задаются специальными константами. Эти константы обычно начинаются с префикса `ad`. Эти константы не включены в программы *Акцент*. Для того чтобы использовать константы, их нужно определить в инструкции `Const`. В комплект поставки *Акцент* входит включаемый файл `ADO.INC`, в котором содержатся наиболее часто используемые константы *ADO*. Включите этот файл в свой модуль с помощью директивы `#include`.

Динамические свойства

Кроме обычных свойств и методов, объекты ADO поддерживают *динамические свойства*. Вид и значения этих свойств зависят от поставщика данных. При подключении к новому поставщику данных эти свойства переопределяются. Динамические свойства реализованы в виде коллекции `Properties`, внутри которой находятся объекты `Property`, описывающие конкретные свойства. Коллекция `Properties` соответствует всем основным объектам *ADO*: `Connection`, `Command` и `Recordset`.

Каждый объект `Property` имеет следующие свойства.

- **`Name`** – имя свойства.
- **`Type`** – целое значение, определяющее тип свойства.
- **`Value`** – непосредственно значение свойства.

В приведенном ниже примере, в электронную таблицу выводятся имена и значения всех динамических свойств текущего соединения (`Connection`).

```
Option Explicit ' обязательные объявления

Dim cn   ' активное соединение
```

```
Dim sht  ' лист таблицы
Dim prop ' объектная переменная
Dim r    '
' получим Connection
Set cn = Workarea.AdoConnection
' создадим таблицу и получим ее лист
Set sht = App.CreateBook.ActiveSheet
i = 1 ' начнем с первой строки

' Для таблицы...
With sht
   .Columns = 2  ' всего две колонки
   ' цикл по всем свойствам Connection
   For Each prop In cn.Properties
      .Cell(i, 1).Value = prop.Name
      .Cell(i, 2).Value = CStr(prop.Value)
      i = i + 1
   Next
   .AutoFit
   .Rows = i-1
End With
```

Объект Connection

Объект `Connection` обеспечивает подключение к источнику данных, а также хранит информацию, связанную с текущим сеансом подключения пользователя к базе данных. Он очень напоминает объект `Database` из *DAO*, но включает и некоторые возможности объекта `Workspace`.

Далее будут рассмотрены основные свойства и методы объекта `Connection`.

Свойство ConnectionTimeout

Целое число, которое определяет продолжительность времени ожидания, в течение которого система будет пытаться установить соединение. Значением свойства является время ожидания в секундах. По умолчанию равно 15. Установка свойства в 0 приведет к тому, что ожидание соединения будет бесконечным.

Свойство Errors

Свойство (только для чтения) возвращает коллекцию ошибок `Errors`. В эту коллекцию добавляются все ошибки, которые произошли с момента подключения к источнику данных.

Свойство Properties

Свойство (только для чтения) возвращает коллекцию динамических свойств `Properties` для соединения (`Connection`).

Метод Open

Начинает сеанс работы с источником данных, открывая сеанс. До выполнения метода `Open` объект нельзя считать активным. В частности, набор свойств (`Properties`) для такого объекта будет описывать только свойства *OLE DB*, связанные с текущим локальным компьютером (такие, например, как идентификатор языка и пр.). Напомним, что свойство `AdoConnection` объекта `Workarea` возвращает ссылку на ОТКРЫТОЕ соединение.

Синтаксис

```
conn.Open([ConnStr],[UserID],[Pwd],[Options])
```

Аргументы

`ConnStr` – строка подключения (*ConnectionString*). Ее синтаксис зависит от поставщика данных. Если не указана, то берется значение свойства `ConnectionString`.

`UserID, Pwd` – имя пользователя. Может быть указано в строке подключения, а может задаваться отдельно.

`Options` – способ подключения. Может быть синхронным или асинхронным. В программах *Акцент* асинхронные подключения не поддерживаются.

Метод Close

Закрывает текущий сеанс работы с источником данных. При вызове этого метода закрывается не только сеанс, но и все другие объекты *ADO* (`Command`, `Recordset`), связанные с этим подключением. Метод не имеет аргументов и не возвращает значений.

Синтаксис

```
conn.Close
```

Метод BeginTrans

Начинает транзакцию. Транзакции могут быть вложенными, если поставщик данных поддерживает эту возможность (*SQL-server* поддерживает). Метод возвращает целое значение, определяющее уровень вложенности вновь начатой транзакции.

Синтаксис

```
conn.BeginTrans
```

Метод CommitTrans

Завершает транзакцию, начатую методом `BeginTrans`. Все изменения транзакции подтверждаются и записываются в базу данных. Отметим, что `CommitTrans` завершает только последнюю из вложенных транзакций. Метод возвращает целое значение, определяющее уровень вложенности завершенной транзакции.

Синтаксис

```
conn.CommitTrans
```

Метод RollbackTrans

Откатывает транзакцию, начатую методом `BeginTrans`. Все изменения, выполненные в рамках текущей транзакции, отменяются, и база данных возвращается в исходное состояние. Отметим, что `RollbackTrans` откатывает только последнюю из вложенных транзакций. Метод возвращает целое значение, определяющее уровень вложенности отмененной транзакции.

Синтаксис

```
conn.RollbackTrans
```

Метод Execute

Хотя, в основном, объект `Connection` обеспечивает управление сеансом работы с базой данных, его можно использовать и для выполнения запросов и получения данных без использования объекта `Command`. Метод `Execute` выполняет SQL-запрос и, возможно, возвращает `Recordset`.

Примечание. Обратите внимание, что передать параметры запросу, выполняемому методом `Execute` объекта `Connection`, невозможно. Если запрос содержит параметры, используйте объект `Command`.

Синтаксис

```
Execute(Command [,ByRef RecsAffected][,Options])
```

Аргументы

`Command` – текст команды. Может быть SQL-запросом, именем таблицы или хранимой процедуры.

`RecsAffected` – переменная, в которую будет записано количество записей, «затронутых» в процессе выполнения команды.

`Options` – тип команды, текст которой записан в `CommandText`. Задается следующими константами:

Константа	Значение	Описание
adCmdText	1	SQL-запрос.
adCmdTable	2	Имя таблицы.
adCmdStoredProc	4	Имя хранимой процедуры.
adCmdUnknown	8	Неизвестно (по умолчанию).

Пример 1 (без возврата `Recordset`)

```
Dim cnn
Dim sql
Set cnn = Workarea.AdoConneciton
sql = "update TABLE set FIELD=1"
cnn.Execute sql
```

Пример 2 (с возвратом `Recordset`)

```
Dim cnn
Dim sql
Dim rs
Set cnn = Workarea.AdoConneciton
sql = "select * from TABLE"
Set rs = cnn.Execute sql
```

Объект Command

Объект `Command` представляет собой команду, которую способен выполнить поставщик данных. В случае работы с *SQL-Server*, команда представляет собой запрос на языке *SQL*.

Команда создается независимо от соединения, а затем устанавливается свойство `ActiveConnection`. Имя объекта команды (для функции `CreateObject`) – `"ADODB.Command"`.

После того как команда создана и подключена к источнику данных, ее можно использовать для выполнения запросов и получения результатов (в виде объекта `Recordset`) от источника данных.

При выполнении запросов с помощью объекта `Command` возможно использование параметров. Для этого используется коллекция `Parameters`, которая содержится в команде. Не забывайте о том, что параметры нужно создать и добавить в коллекцию до того, как команду можно будет выполнить.

Далее будут рассмотрены основные свойства и методы объекта `Connection`.

Свойство CommandTimeout

Целое число, которое определяет продолжительность времени ожидания, в течение которого система будет ожидать завершения выполнения команды. Значением свойства является время ожидания в секундах. По умолчанию равно 30. Установка свойства в 0 приведет к тому, что ожидание выполнения команд будет бесконечным.

Свойство ActiveConnection

Свойство определяет активное соединение, в контексте которого будет выполняться команда. Свойство должно быть установлено перед использованием любых методов команды. Свойство представлено объектом, поэтому указание ключевого слова `Set` при работе с ним обязательно.

Синтаксис

```
cmd.ActiveConnection
```

Пример

```
Dim cnn
Dim cmd
Set cnn = Workarea.AdoConneciton
Set cmd = CreateObject("ADODB.Command")
' свяжем Command с Connection
Set cmd.ActiveConnection = cnn
```

Свойство CommandText

Текстовая строка, определяющая текст команды. Ее синтаксис зависит от поставщика данных.

Свойство CommandType

Целое число, определяющее тип команды, текст которой задан свойством `CommandText`. Если тип команды не задан, то источник данных сам пытается определить, что за команду он получил. Естественно, что на это требуется время. Поэтому, в целях увеличения скорости обработки лучше задавать тип команды явно.

Тип команды задается одной из следующих констант.

Константа	Значение	Описание
`adCmdText`	1	SQL-запрос.
`adCmdTable`	2	Имя таблицы.
`adCmdStoredProc`	4	Имя хранимой процедуры
`adCmdUnknown`	8	Неизвестный тип команды (по умолчанию)

Свойство Properties

Свойство (только для чтения) возвращает коллекцию динамических свойств `Properties` для команды (`Command`).

Свойство Parameters

Свойство возвращает коллекцию параметров, которые будут использованы при выполнении запроса. Перед тем, как параметризованный запрос будет выполнен, все параметры должны быть созданы (методом `CreateParameter`) и добавлены в коллекцию `Parameters`.

Свойство Prepared

Свойство устанавливает признак того, что команда перед выполнением должна быть «подготовлена», и затем сохранена в таком состоянии. Это свойство очень полезно в случае, если нужно выполнить много одинаковых запросов, которые отличаются только значениями параметров.

Метод CreateParemeter

Метод создает новый параметр (объект `Parameter`) и возвращает его. Метод НЕ добавляет вновь созданный параметр в коллекцию `Parameters`. Это нужно сделать методом `Append` коллекции `Parameters`.

Синтаксис

```
CreateParameter(Name,Type,Dir,Size,Value)
```

Аргументы

`Name` – имя параметра. Обычно параметры связываются не по именам, а по порядковым номерам. В этом случае имя должно быть пустым.

`Type` – тип параметра. Задается константами типов данных. Подробности в справочной системе *ADO*.

`Dir` – направление передачи данных. Параметр может быть входным, выходным, двунаправленным или принимать возвращаемое хранимой процедурой значение.

`Size` – размер параметра. Имеет смысл для строковых данных.

`Value` – значение параметра.

Метод Execute

Основной метод объекта `Command`. Он выполняет запрос и, возможно возвращает результат в виде объекта `Recordset`.

Синтаксис

```
Execute([ByRef RecsAffected][,Params][,Options])
```

Аргументы

`RecsAffected` – переменная, в которую будет записано количество записей, «затронутых» в процессе выполнения команды.

`Params` – коллекция параметров, которую можно передать в метод. Если она не указана, то используется внутренняя коллекция параметров (возвращаемая свойством `Parameters`)

`Options` – тип команды. Это альтернативный свойству CommandType способ задания типа команды.

Пример 1 (без возврата `Recordset`)

```
Dim cnn
Dim cmd
Set cnn = Workarea.AdoConneciton
Set cmd = CreateObject("ADODB.Command")
Set cmd.ActiveConnection = cnn
cmd.CommandText = "update TABLE set FIELD=1"
cmd.CommandType = adCmdText
cmd.Execute
```

Пример 2 (с возвратом `Recordset`)

```
Dim cnn
```

```
Dim cmd
Dim rs
Set cnn = Workarea.AdoConneciton
Set cmd = CreateObject("ADODB.Command")
Set cmd.ActiveConnection = cnn
cmd.CommandText = "select * from TABLE"
cmd.CommandType = adCmdText
Set rs = cmd.Execute
```

Объект Recordset

Объект `Recordset`, без сомнения, представляет собой основу объектной модели *ADO* и является самым большим и сложным.

Объект `Recordset` тесно связан с понятием *курсора*. Курсор это некая сущность, напоминающая плоскую таблицу или массив со строками (записями) и столбцами (полями). Когда создается объект `Recordset`, то фактически создается курсор, а затем он заполняется данными, полученными от поставщика данных. Курсоры бывают разных видов.

Самый простой из них, так называемый *forward only*, используется только для чтения данных, причем перемещение по строкам возможно только вперед. Этот вид курсора используется наиболее часто и затрачивает наименьшее количество системных ресурсов. Рекомендуется всегда использовать такие курсоры, если нет веских причин для использования других типов курсоров.

Следующий вид курсора – статический (*static*). Как и предыдущий вид курсора, он не допускает изменений данных, но позволяет перемещаться по строкам в обоих направлениях.

Динамические (*dinamic*) курсоры самые сложные. Они допускают изменения данных, перемещение в обоих направлениях, синхронизацию изменений, сделанных другими пользователями. Такие курсоры используют очень много ресурсов. Поэтому их следует применять только в случае крайней необходимости.

Помимо типа курсора имеет значение и его местоположение. Многие поставщики данных (в том числе *Microsoft SQL-Server*) допускают создание курсоров, как на стороне сервера, так и на стороне клиента. Настоятельно не рекомендуется использовать курсоры на стороне клиента, если для этого нет очень веских причин. Такие курсоры могут значительно снизить производительность и резко увеличивают сетевой трафик.

Одна из строк курсора всегда является текущей. Именно для нее будут выбираться или устанавливаться данные.

Столбцы (поля) курсора представлены коллекцией `Fields`. Каждый элемент этой коллекции (объект `Field`) представляет собой одно поле данных.

Далее будут перечислены основные свойства и методы `Recordset`.

Свойство CursorLocation

Свойство позволяет задать местоположение курсора и может принимать одно из следующих значений

- **`adoUseClient`** – курсор создается на клиенте
- **`adoUseServer`** – курсор создается на сервере. (Это значение используется по умолчанию).

Объект `Recordset` автоматически наследует значение этого свойства из активного соединения (`Connection`). Обратите внимание, что если `Recordset` открыт, свойство доступно только для чтения.

Свойство CursorType

Свойство позволяет задать тип курсора и может принимать одно из следующих значений.

- **`adoOpenForwradOnly`** – курсор *forward only*. (по умолчанию)
- **`adoOpenStatic`** – статический курсор.
- **`adoOpenDynamic`** – динамический курсор.

Свойства курсоров различных типов описаны выше.

Свойства EOF, BOF

Эти свойства используются для навигации по записям. Свойство `EOF` становится равным `True`, если текущая запись находится непосредственно за последней записью набора. Свойство `BOF`, наоборот, говорит о том, что текущая запись находится перед первой записью набора данных.

Обычно эти свойства используются внутри цикла `While`, который перебирает записи в наборе.

Примеры

```
' проход по всем записям набора
While Not rs.EOF
   ' что-то делаем
   rs.MoveNext
Wend

' проверка набора на пустоту
If rs.EOF And rs.BOF Then
   MsgBox "Нет записей"
End If
```

Свойство RecordCount

Очень «опасное» свойство. Оно возвращает количество записей в наборе. Однако, очень часто определить это количество можно только тогда, когда все данные набора уже прочитаны. Это может привести к резкому падению производительности. Используйте это свойство только в случае крайней необходимости. На практике не существует случаев, когда то же самое нельзя сделать каким-либо другим способом.

Свойство MaxRecords

Это свойство позволяет ограничить максимальное число записей, возвращаемых при выполнении запроса. Очень полезно при выполнении незапланированных запросов, чтобы ограничить нагрузку на сервер.

Свойство Fields

Это свойство возвращает коллекцию полей данных, представляющих текущую запись в наборе.

Метод Open

Этот метод открывает `Recordset`. Он имеет множество аргументов, позволяющих задать соединение, команду, тип курсора и способ блокировки записей в наборе.

Однако чаще всего объект `Recordset` возвращается уже открытым (методом `Execute` объектов `Command` или `Connection`), поэтому мы не будем описывать метод `Open`. За подробными сведениями обращайтесь к документации по *ADO*.

Методы MoveNext, MovePrevious, MoveFirst, MoveLast

Эти методы перемещают указатель текущей записи на одну запись вперед (`MoveNext`), на одну запись назад (`MovePrevious`), на первую запись (`MoveFirst`) и на последнюю запись (`MoveLast`). Очевидно, что для курсоров типа *forward only*, попытка выполнить методы `MovePrevious` и `MoveFirst` приведет к ошибке.

При работе с этими методами всегда нужно анализировать положение текущей записи (свойства `EOF` и `BOF`). Попытка перемещения за пределы набора приведет к ошибкам выполнения программы.

Метод NextRecordset

Некоторые хранимые процедуры могут возвращать не один, а несколько наборов данных. Для того чтобы переключиться на следующий набор данных, используется метод `NextRecordset`. Обратите внимание, что переключиться назад, на предыдущий набор невозможно.

Метод Requery

Метод `Requery` повторно выполняет команду, с помощью которой был создан объект `Recordset`. При этом в набор попадут все изменения, которые произошли с данными, с момента последнего выполнения команды. Фактически, выполнение метода `Requery` равнозначно закрытию и повторному открытию набора.

Объект Field и коллекция Fields

Коллекция `Fields` представляет собой набор полей данных для текущей записи `Recordset`. Как и все коллекции, она очень проста и имеет всего два свойства. Свойство `Count` возвращает количество элементов в коллекции, а свойство `Item` позволяет получить конкретный объект `Field` по имени или по индексу.

Объект `Field` более интересен и имеет следующие свойства.

- **`Name`** – имя поля.
- **`Type`** – тип данных, которые хранятся в поле. Конкретные типы определяются специальными константами. Их можно посмотреть в файле `ADO.INC` из комплекта поставки.

- **Value** – значение поля (тип Variant). Свойство является свойством «по умолчанию» и его можно опускать.
- **DefinedSize** – длина поля, как она определена в базе данных. Имеет смысл для полей типа varchar и text.
- **ActualSize** – реальная длина данных, которые хранятся в поле. Имеет смысл для полей типа varchar и text.

Пример

```
Dim f     ' объектная переменная для поля
Dim name  ' имя поля
Dim value ' значение поля
' проход по всем записям набора
While Not rs.EOF
   For Each f In rs.Fields
      name = f.Name
      value = f.Value
      ' что-то делаем
   Next
   rs.MoveNext
Wend
```

Глава XII.

Дополнительные сведения

Обработка ошибок и объект Err

Язык *VBScript* имеет простейшие возможности обработки ошибок, возникающих во время выполнения программы. Если обработка ошибок включена, то при возникновении исключительных ситуаций программист имеет возможность «вмешаться» в процесс до того, как появится стандартное сообщение об ошибке и программа будет остановлена.

Объект `Err` содержит полное описание возникшей ошибки. Он имеет следующие свойства (все они только для чтения).

- **`Description`** – текстовое описание ошибки.
- **`HelpFile`** – полный путь к файлу справки, которые содержит описание данной ошибки.
- **`HelpContext`** – номер раздела справочной системы, содержащий информацию об ошибке.
- **`Number`** – внутренний код ошибки.
- **`Source`** – имя объекта, ставшего источником ошибки.

Для того чтобы ошибка была перехвачена, обязательно должна быть выполнена инструкция `On Error`. Если она не была выполнена, любая ошибка времени выполнения приведет к выдаче сообщения об ошибке и завершению работы программы. Если инструкция `On Error Resume Next` была выполнена, то с этого места и до завершения области видимости (обычно до завершения процедуры или функции), то появление ошибки приводит к тому, что выполнение программы будет продолжено.

Для того чтобы проверить, имела ли место ошибка, нужно проанализировать значение свойства `Number` стандартного объекта `Err`. Если оно ненулевое, то имела место ошибка. Сообщение об ошибке можно получить из свойства `Description` объекта `Err`. После того как ошибка обработана, нужно очистить ее вызвав метод `Clear`.

Пример обработки ошибок

В следующей программе показано, как обрабатываются ошибки времени выполнения (Run-time errors).

```
' включим обработку ошибок
On Error Resume Next

Dim x ' просто переменная

x = 1 / 0 ' деление на нуль вызовет ошибку

' Обработка включена, значит If выполнится
If Err.Number <> 0 Then
  ' выведем собственное сообщение
  MsgBox "Error # " & _
         CStr(Err.Number) & " " & _
         Err.Description
  Err.Clear ' Очистить ошибку
End If
```

Классы в VBScript

Класс в *VBScript* это описание объекта, создаваемого пользователем. Пользовательский объект такой же, как и встроенный объект системы: он хранит в себе данные, имеет свойства для доступа к этим данным и методы для выполнения каких-либо действий. Общее название для свойств и методов - *члены класса*.

Класс объявляется инструкцией Class, а конкретный экземпляр класса создается инструкцией New.

Инструкция Class

Элементы данных, свойства и методы класса могут быть объявлены как общедоступные (Public) или закрытые (Private). Все, что описано как Private, видимо только внутри класса.

Если для описания данных используется инструкция Dim, то элементы считаются общедоступными (Public).

Свойства класса описываются инструкциями Property Get, Property Let, Property Set, а методы похожи на обычные процедуры и функции и описываются инструкциями Sub и Function.

Одно свойство (или один метод) класса может быть помечено как свойство «по умолчанию». Это делается с помощью ключевого слова `Default`. Свойство «по умолчанию» автоматически используется при упоминании имени класса без указания свойства.

Существует два предопределенных метода класса. Первый из них, обычно называемый *«конструктор»*, автоматически выполняется при создании экземпляра класса и чаще всего служит для начальной инициализации переменных класса. Имя метода – `Class_Initialize`.

Второй метод, называемый *«деструктор»*, автоматически выполняется при уничтожении экземпляра класса. Он имеет имя `Class_Terminate`.

Инструкции Property Get и Property Let

Эти инструкции определяют имя, аргументы и тело функции, которая будет использоваться для чтения значения или записи свойства класса. Инструкции `Property Get` и `Property Let` всегда находятся внутри определения класса, то есть внутри инструкции `Class`. Инструкции выглядят как обычные описания функций и завершаются парой ключевых слов `End Property`.

Свойства могут быть помечены как закрытые (`Private`). Закрытое свойство может использоваться только внутри класса. Зарытые свойства помечаются ключевым словом Private, которое записывается перед началом определения свойства.

Инструкция `Exit Property` приводит к немедленному выходу из функции чтения или записи свойства. Можно использовать любое количество вызовов `Exit Property`.
Для того чтобы вернуть значение свойства, необходимо присвоить значение псевдопеременной, имя которой совпадает с именем свойства.

Инструкция Property Set

Если свойство должно возвратить ссылку на объект, то вместо инструкции `Property Let` нужно использовать `Property Set`. Такое различие связано с понятием «свойство по умолчанию».

Работа с внешними приложениями

Язык *VBScript* полностью поддерживает технологию *OLE-Automation*. Это означает, что Вы можете управлять любыми объектами, компонентами и приложениями, которые поддерживают этот метод управления.

В частности, все приложения семейства *Microsoft Office* полностью поддерживают технологию *OLE-Automation*. Это означает, что Вы можете полностью интегрировать эти приложения в свою рабочую среду.

Основные встроенные функции, которые отвечают за работу с внешними приложениями, это `CreateObject` и `GetObject`.

Функция `CreateObject` создает *ActiveX* объект по имени (или по GUID) сервера. Например, создать объект *Microsoft Excel* можно с помощью следующего фрагмента кода:

```
' создание рабочей книги Excel
Dim sh

Set sh = CreateObject("Excel.Workbook")
' теперь можно использовать все свойства
' и методы Excel.
```

После того как объект успешно создан (значение объектной переменной не равно `Nothing`), можно использовать все свойства и методы этого объекта. Для того чтобы изучить объектную модель внешнего приложения можно воспользоваться документацией, поставляемой с этим приложением, а также дополнительными инструментами, например *OLE/COM Object Viewer*, который поставляется в составе инструментов разработчика *Accent 7.4 Developer Kit*.

HTML и Акцент

Программа *Акцент 7.4* имеет развитые средства работы с языком *HTML*. Показ HTML-страниц выполняется средствами браузера *Internet Explorer*.

Программы на языке *VBScript*, из которых вызываются HTML-страницы имеют полный доступ к объектной модели страницы (*DOM-Document Object Model*). Другими словами, из программы на *VBScript* можно изменять содержимое тегов страницы, вызывать обработчики событий и т.д.

Обратное тоже верно. Любой программный код, который содержится на HTML-странице (на языках *VBScript* или *JScript*) имеет доступ к объектной модели *Акцента*.

В программе существует четыре основных части, в которых возможно использование HTML.

- **Web-интерфейс** – существует возможность полностью заменить интерфейс пользователя *Акцент* на собственный, написанный на языке HTML.
- **Ассистент** – это панель управления в правой части основного окна программы, формы или электронной таблицы, в которой может показываться динамический HTML-текст.
- **DHTML-диалог** – специальный объект, который позволяет показать содержимое HTML-страницы в специальном диалоговом окне (как модальном, так и немодальном).
- **Окно броузера** – в приложении *Акцент* можно открыть окно браузера, в котором будет отображаться произвольная HTML-страница. При этом скрипты, которые содержатся на странице имеют доступ к объектной модели *Акцент*.

Доступ из VBScript к DOM

Доступ из *VBScript* к объектной модели *DOM* показываемой HTML-страницы наиболее прост. Для этого используется свойство `HTMLDocument`. Это свойство имеется у всех объектов, показывающих HTML-страницы.

Свойство возвращает ссылку на объект `document` объектной модели DOM (*Document Object Model*). Объектная модель DOM достаточно сложна и не описана в этой книге. Для более подробного изучения воспользуйтесь документацией корпорации *Microsoft* или дополнительной литературой.

Доступ из HTML к объектной модели Акцент

Для доступа к объектной модели *Акцент* из скриптов, содержащихся на HTML-страницах используется специальная особенность DOM, которая предоставляет возможность доступа к объектной модели хост-приложения. В нашем случае хост-приложением является *Акцент*.

Для такого доступа используется свойство `external` объекта `window` объектной модели DOM. Это свойство возвращает ссылку на объект, изнутри которого запущен HTML-документ. В таблице ниже приведены объекты, возвращаемые свойством `external`.

Элемент системы	Возвращаемый объект
Web-интерфейс	Workarea
Ассистент (основное окно)	Workarea
Ассистент (форма)	Form
Ассистент (таблица)	ShtBook
DHTMLDialog	Workarea
Окно браузера	Workarea

DHTML-диалоги

DHTML-диалоги позволяют показать HTML-страницу в диалоговом окне. Окно может быть как модальным, так и немодальным. Помимо этого при работе с DHTML-диалогом имеется возможность размещать на нем командные кнопки и определять из основной программы, какая кнопка была нажата пользователем.

DHTML-диалоги управляются с помощью объекта `DHTMLDialog`.

Более подробная информация о работе с DHTML-диалогами приведена в электронной справочной системе.

Web-интерфейс

В программе Акцент имеется возможность полностью заменить интерфейс пользователя на собственный, построенный на основе HTML-страниц. В этом случае интерфейс основного окна программы полностью скрывается от пользователя, а вместо этого основное окно программы заменяется браузером.

Для вызова такого интерфейса можно воспользоваться командой «**Начальная страница**» из меню «**Вид**». В опциях настройки приложения имеется возможность указать, что программа всегда должна запускаться под управлением Web-интерфейса.

Как и в предыдущих случаях, скрипты на HTML-странице Web-интерфейса имеют полный доступ к объектной модели *Акцент* через свойство `external` объекта `window` объектной модели DOM. В этом случае свойство `external` возвращает ссылку на объект `Workarea`.

Обратите внимание, что Web-интерфейс может отображаться и тогда, когда база данных не открыта. Объект `Workarea` в этом случает тоже существует, но ссылается на неоткрытую базу данных (свойство `IsDatabaseOpen` равно `False`). Открыть базу данных можно с помощью метода `OpenDatabase`.

Встроенный браузер

Встроенный в приложение браузер (фактически это ядро *Microsoft Internet Explorer*) может отображаться в отдельном окне.

Для вызова такого окна используется метод `OpenURL` объекта `Application`. В качестве аргумента в метод передается URL заданной страницы. Это может быть как http-адрес, так и имя локального файла на диске пользователя, причем, если это файл, то при его поиске используется информация о расположении файлов (из файла `*.ard`).

Обратите внимание на то, что страница, открытая в окне браузера также имеет доступ к объектной модели *Акцент*. В этом случае свойство `external` объекта `window` объектной модели DOM возвращает ссылку на объект `Workarea`.

Приложения

В приложениях приведен список ключевых слов языка программирования VBScript, описаны стандартные объекты языка, такие как `Font`, `Picture`, `Err` и часто используемые объекты среды *Windows Scripting Host*, такие как объект файловой системы – `FileSystemObject` и объект для работы с текстовыми файлами – `TextStream`.

A. Список ключевых слов VBScript

And	Eqv	Loop	ReDim
ByRef	Erase	Me	Rem
ByVal	Error	Mod	Resume
Call	Execute	New	Select
Case	Exit	Next	Set
Class	Explicit	Not	Step
Const	For	On	Sub
Default	Function	Option	Then
Dim	Get	Or	To
Do	If	Preserve	Until
Each	Imp	Private	Wend
Else	In	Property	While
ElseIf	Is	Public	With
End	Let	Randomize	Xor

B. Стандартные объекты VBScript

Объект Font

Объект `Font` представляет собой шрифт *Windows*. Он используется там, где нужно работать со шрифтами, устанавливать их размер, стиль, начертание и т.д. Если какой-либо объект или элемент управления имеет свойство `Font`, то оно возвращает ссылку именно на этот объект.

Основные свойства объекта `Font` следующие:

- **Name** – имя семейства шрифта, например «*Arial*».
- **Size** – размер шрифта. Задается в пунктах. Свойство имеет тип `Currency`, что позволяет задавать дробные значения размера с точностью 0.0001 pt.
- **Bold** – булево значение. Если оно равно `True`, то это полужирный шрифт.
- **Italic** – булево значение. Если оно равно `True`, то это курсивный шрифт.
- **Underline,Strikethrough** – подчеркивание или перечеркивание. Если значение свойства равно True, то текст будет выводиться либо перечеркнутым (`Strikethrough`), либо подчеркнутым (`Underline`).
- **Weight** – жирность шрифта. В отличие от свойства `Bold`, которое позволяет задать либо светлый либо полужирный шрифт, это свойство управляет жирностью более гибко. Диапазон возможных значений от 100 (очень тонкий шрифт) до 900 (очень жирный шрифт). Жирность обычного шрифта равна 400, а полужирного – 700.
- **Charset** – целое значение, которое задает набор символов (кодовую страницу) для шрифта. Значение для русского языка равно 204. Обратите внимание, что некоторые шрифты заранее связаны с конкретным набором символов. Если, например, указать шрифт «`Arial Cyr`», то он уже будет иметь значение свойства `Charset` равное 204.

Объект Picture

Объект `Picture` обеспечивает доступ к файлам изображений. Если у объекта имеется свойство `Picture`, то оно возвращает ссылку именно на этот объект.

В *VBScript* имеется встроенная функция `LoadPicture`, которая загружает картинку из файла и возвращает ссылку на объект `Picture`. Такую ссылку можно присвоить свойству `Picture` и тем самым связать картинку с объектом *Акцент*.

Объект имеет достаточно много методов и свойств, однако, в программах Акцент реально используются только некоторые из них:

- **Width**, **Height** – задают ширину (Width) и высоту (Height) изображения. Значения задаются в сотых долях миллиметра (0.01 мм).
- **Type** – целое число, характеризующее тип картинки. Например, для растровой картинки это значение равно 1, а для значка (иконки) – 3. Полный перечень типов содержится в электронной справочной системе.

Объект Dictionary

Объект Dictionary представляет собой набор пар «имя-значение». В программировании такая структура данных известна как *хэш-таблица* или *ассоциативный массив*. Основное его отличие от обычного массива в том, что он обеспечивает очень быстрый доступ к значениям по ключу.

Свойства и методы объекта **Dictionary** следующие.

- **Item** – свойство доступно для чтения и записи и устанавливает значение ключа. Ключ выступает в качестве аргумента свойства.
- **Count** – свойство только для чтения и возвращает количество элементов в словаре.
- **Exists** – метод возвращает True, если заданный ключ имеется в словаре.
- **Keys** – свойство только для чтения, которое возвращает все значения ключей, которые хранятся в словаре. Обратите внимание, что порядок, в котором ключи попадут в массив, не определен.
- **Add** – добавляет в словарь пару ключ-значение. Если ключ уже существует, происходит ошибка. Именно этим метод отличается от свойства Item, которое в случае дублирования значений ключа просто заменяет старое значение новым.
- **Remove** – удаляет ключ из словаря. Ключ задается как аргумент свойства.
- **RemoveAll** – полностью очищает словарь, удаляя из него все значения. После этого свойство Count будет равно нулю.

C. Стандартные Объекты WSH

Объекты FileSystemObject и TextStream

Объект `FileSystemObject` используется для доступа к файловой системе локального компьютера. Для работы с текстовыми файлами служит объект `TextStream`. Эти объекты входят в состав библиотеки *Windows Scripting Host*, которая поставляется вместе с *VBScript*.

Создание объекта возможно только при помощи инструкции `CreateObject` с именем `"Scripting.FileSystemObject"`.

Объект содержит довольно много методов и свойств, позволяющих делать с файловой системой практически любые действия. Здесь мы рассмотрим только некоторые из них, связанные с обработкой текстовых файлов.

- **`CreateTextFile`** – метод создает текстовый файл с заданным именем и возвращает ссылку на объект `TextStream`. Помимо имени, метод имеет аргументы, позволяющий указать, нужно ли перезаписывать файл, если он существует, а также задать формат файла (*ANSI/Unicode*).
- **`OpenTextFile`** – метод открывает существующий файл или создает новый в случае необходимости. Метод возвращает ссылку на объект `TextStream`. Дополнительные аргументы позволяют задать режим открытия файла (чтение/запись/добавление), поведение метода, в случае, если файл не существует и формат файла (*ANSI/Unicode*).

Объект `TextStream` представляет собой потоковый текстовый файл. Файл читается и записывается построчно. Объект имеет следующие свойства и методы.

- **`Line`**, **`Column`** – номер строки и позиция в текущей строке. Позиция в строке имеет смысл только для посимвольного чтения файла.
- **`AtEndOfLine`** – свойство возвращает `True`, если указатель чтения файла находится непосредственно перед символом конца строки. Свойство применимо только к файлам, которые открыты «только для чтения».
- **`AtEndOfStream`** – свойство возвращает `True`, если указатель чтения файла находится в конце файла и далее читать нечего. Свойство применимо только к файлам, которые открыты только для чтения.

- **Read** – метод читает заданное количество символов из файла.
- **ReadLine** – метод считывает из файла одну строку (до символа конца строки).
- **ReadAll** – метод считывает весь файл и возвращает его в виде текстовой строки.
- **Write** – записывает в файл текстовую строку.
- **WriteLine** – записывает в файл текстовую строку и завершает ее символом конца строки.
- **WriteBlankLines** – записывает в файл один или несколько символов перевода строки.
- **Skip** – метод пропускает заданное число символов при чтении из файла.
- **SkipLine** – метод пропускает строку при чтении из файла.

Пример работы с файловой системой

Ниже приведен пример программы, которая записывает в текстовый файл строку, а затем ее читает.

```
Option Explicit

' константы для открытия файла
Const ForReading   = 1 ' чтение
Const ForWriting   = 2 ' запись
Const ForAppending = 3 ' добавление

' имя файла
Const FileName = "c:\textfile.txt"

Dim fs ' объект FileSystemObject
Dim f  ' объект TextStream
Dim s  ' строка для чтения

' создать объект FileSystemObject
Set fs = _
CreateObject("Scripting.FileSystemObject")
' создадим файл. Если файл уже есть,
' то он перезапишется
```

```
Set f = fs.CreateTextFile(FileName,True)
' запишем в него строку
f.Write "Это строка текста!"
' закроем файл
f.Close
' уничтожим объект
Set f = Nothing

' откроем файл для чтения
Set f = fs.OpenTextFile(FileName, ForReading)
' прочитаем из него строку
s = f.ReadLine
' выведем строку на экран
MsgBox s
' закроем файл
f.Close
' уничтожим объект
Set f = Nothing
```

Алфавитный указатель

A

Abs(), функция 54
Account, объект..... 118
Accounts, коллекция..... 118
ActiveConnection, свойство (ADO) 255
ADO..... 248
AdoConnection, свойство..... 249
AfterSave, событие..... 126
Agent, объект 119
Agents, коллекция..... 119
App(), функция..... 88, 94
Application, объект 93
Array(), функция..... 33, 56
Asc(), функция 62
ASCII-код 62, 70
Ask(), функция 85

B

BeforeSave, событие..... 126
BeginTrans, метод (ADO)..... 253
Binder, объект..... 121
Binders, коллекция..... 121
BOF, свойство (ADO)..... 260

C

Call, инструкция 47
CBool(), функция 50
CByte(), функция..... 50
CCur(), функция 50
CDate(), функция..... 51
CDbl(), функция..... 52
CharAt(), функция..... 71
CheckNull(), функция 88
Chr(), функция 62
CInt(), функция 52
Class, инструкция..... 264
CLng(), функция..... 52
Close, метод (ADO) 253
Command, объект (ADO)..... 255
CommandText, свойство (ADO) 256
CommandTimeout, свойство (ADO)..... 255
CommandType, свойство (ADO) 256
CommitTrans, метод (ADO)... 253
Connection, объект (ADO)..... 251
ConnectionTimeout, свойство (ADO)..... 252
Const, инструкция 27
CreateBook, метод 224
CreateDialog, метод 171, 196
CreateObject(), функция 58
CreateParameter, метод (ADO) 257
CreateReport, метод..... 142
CSng(), функция..... 52
CStr(), функция 62
CursorLocation, свойство (ADO) 259
CursorType, свойство (ADO). 259

D

DAO 233
DaoDatabase, свойство.. 235, 238
DaoEngine, свойство 235
Database, объект (DAO)..... 238
Date(), функция..... 75
DateAdd(), функция 75
DateDiff(), функция..... 76

DatePart(), функция.....77
DateSerial(), функция.....77
DateValue(), функция.....78
Day(), функция.....78
DbEngine, объект (DAO).....236
DHTMAssistant, объект.....139
DHTMLDialog, объект.....137
Dictionary, объект.....273
Dim, инструкция.....26, 32
DlgButton, элемент диалога...204
DlgCheck, элемент диалога...205
DlgComboBox, элемент диалога
.....208
DlgDate, элемент диалога.....207
DlgEdit, элемент диалога.....206
DlgFrame, элемент диалога...203
DlgGrid, элемент диалога.....209
DlgGridRange, диапазон ячеек таблицы.....212
DlgGroupBox, элемент диалога
.....203
DlgLabel, элемент диалога....203
DlgListBox, элемент диалога.207
*DlgListView, элемент диалога*215
DlgMeter, объект.....133
DlgMeter, элемент диалога....209
DlgPictire, элемент диалога..204
DlgRadio, элемент диалога....205
DlgSelector, элемент диалога 207
*DlgTabStrip, элемент диалога*214
DlgTreeView, элемент диалога
.....218
Do…Loop, инструкция.....42
DoEvent(), функция.....88
Dynamic HTML.....266

E

Entities, коллекция.....120
Entity, объект.....120
EntUnit, объект.....120
EntUnits, коллекция.....120
EOF, свойство (ADO)..... 260
Erase, инструкция..... 34
Err, объект..... 263
Error, объект (DAO)..... 237
Errors, свойство (ADO)..... 252
Execute, инструкция 48
Execute, метод (ADO) ... 254, 257
ExecuteGlobal, инструкция 48
Exit Do, инструкция..... 43
Exit For, инструкция..... 41
Exit Function, инструкция 47

F

Fact, объект 123
Facts, коллекция 123
Field, объект (ADO) 261
Field, объект (DAO) 240
Fields, коллекция (DAO)..... 240
Fields, коллекция(ADO)..... 261
Fields, свойство (ADO) 260
FileDialog, объект..... 134
FileSystemObject, объект 274
Filter(), функция..... 64
Fix(), функция..... 52, 54
Font, объект..... 271
For…Each, инструкция 44
For…Next, инструкция 41
Form, объект..... 172
FormatDate2(), функция..... 81
FormatDateTime(), функция..... 79
FrmButton, элемент формы.. 183
FrmCheck, элемент формы... 183
FrmComboBox, элемент формы
..... 185
FrmDate, элемент формы..... 185
FrmEdit, элемент формы..... 184
FrmGrid, элемент формы..... 186
FrmGridRange, объект..... 188
FrmLabel, элемент формы.... 180
FrmLine, элемент формы..... 181
FrmPage, объект 175

FrmPicture, элемент формы . 182
FrmRadio, элемент формы ... 183
FrmRect, элемент формы..... 183
FrmScrollGrid, элемент формы 189
FrmSelector, элемент формы 184
FrmSGridCell, объект..... 193
FrmSGridColumn, объект..... 192
Function, инструкция..... 47

G

GetObject(), функция..... 59
GetRef(), функция 86

H

Hex(), функция..... 65
Hour(), функция..... 80
HTML 137, 139, 266

I

If…Then…Else, инструкция..... 39
IIF(), функция 87
InputBox(), функция..... 85
InStr(), функция 65
InStrRev(), функция 65
Int(), функция..... 52, 54
IsArray(), функция 57
IsDate(), функция..... 51, 80
IsFileExists(), функция 91
IsNumeric(), функция..... 52
IsObject(), функция..... 58

J

Join(), функция 66

L

LBound(), функция..... 33, 57
LCase(), функция 66
Left(), функция 67
Len(), функция 67
LoadPicture(), функция 60
LoadPicture, функция 272
LTrim(), функция..... 63

M

MakeSheet, метод..... 224
Map, объект..... 22, 127
MaxRecords, свойство (ADO) 260
Meter(), функция..... 89, 133
Meter, объект 133
Mid(), функция 67
Minute(), функция 80
Month(), функция 78
MoveFirst, метод (ADO)..... 261
MoveLast, метод (ADO)..... 261
MoveNext, метод (ADO) 261
MovePrevious, метод (ADO) . 261
MsgBox(), функция 83

N

NextRecordset, метод (ADO) . 261
Now(), функция 75

O

Oct(), функция..... 65
OnApply, событие 126
OnCollapse, событие 97
OnEdit, событие..... 126
OnExpand, событие 97
OnPopup, событие 134
OnRecalc, событие..... 126
Open, метод (ADO)..... 252, 261
Operation, объект..... 105, 106
Option Explicit, инструкция26, 32

P

PageSetup, объект..... 130
Param, объект 122
Parameter, объект (DAO)..... 245
Parameters, коллекция (DAO) 245
Parameters, свойство (ADO) . 256
Params, коллекция 122

Period, объект 129
Picture, объект 272
PopupMenu, объект.....133
PopupToolTip, объект 137
Prepared, свойство (ADO).....256
Properties, свойство (ADO) ..252, 256
Property Get, инструкция.....265
Property Let, инструкция 265
Property Set, инструкция.....265

Q

QueryDef, объект (DAO).....244
QuickSort(), функция.....89

R

RBound(), функция.....33
RecordCount, свойство (ADO) 260
Recordset, объект (ADO) 258
Recordset, объект (DAO) 239
ReDim, инструкция.....33, 34
Redirect, объект.....129
RepBalance, отчет.....145
RepBalanceItem, объект.....146
RepBankList, отчет.....151
RepBankListItem, объект.....152
RepCashBook, отчет.....153
RepCashBookItem, объект.....154
RepChess, отчет.....144
RepCommon, отчет.....143
RepCross, отчет.....146
RepDocList, отчет.....149
RepDocListItem, объект 151
RepGLedger, отчет.....157
RepJournal, отчет.....158
RepJournalItem, объект.....160
Replace(), функция.....68
RepSubList, отчет.....147
RepSubListItem, объект.....149
RepSynth, отчет 155
RepTax, отчет 166
RepTurn, отчет..... 156
RepWizard, отчет..... 161
RepWizardItem, объект..... 165
Requery, метод (ADO)..... 261
Right(), функция..... 67
RollbackTrans, метод (ADO). 253
Round(), функция..... 55
Round2(), функция..... 55
RTrim(), функция..... 63

S

Second(), функция..... 80
Select…Case, инструкция..... 40
Set, инструкция..... 35
Sgn(), функция..... 54
ShtBook, объект 225
ShtRange, объект 230
ShtSheet, объект..... 228
Space(), функция..... 68
Spell(), функция..... 71
SpellMoney(), функция 72
SpellMoney2(), функция 72
Split(), функция..... 69
StrComp(), функция..... 69
String(), функция 70
StrReverse(), функция..... 70
Sub,инструкция..... 46
System, объект 135

T

TaxReport, объект..... 125
TaxReports, коллекция..... 125
Template, объект..... 121
TemplateInvoker, объект..... 126
Templates, коллекция..... 121
TextStream, объект 274
Time(), функция 75
TimeSerial(), функция..... 81
Token(), функция 73
Toolbar, объект..... 131

ToolbarButton, объект 131
Transaction, объект 105, 113
TransList, объект 105, 111
Trim(), функция 63

U

UBound(), функция 57
UCase(), функция 66

V

VarType(), функция 53

W

WaitCursor(), функция 89, 132
WaitCursor, объект 132
While…Wend, инструкция 45
WinAPI, объект 136
Workarea(), функция 91, 99
Workarea, модуль 21
Workarea, объект 99
Workspace, объект (DAO) 237

Y

Year(), функция 78

К

Класс 27
Комментарии 23
Константа 27

Л

Литерал 24

М

Массив 33
  ассоциативный 22, 273
  динамический 33
  открытый 33
Метод 29

П

Панель управления 131
Переменная 26

С

Событие 29

Ш

Шрифт 271
  набор символов 272